June 92

NEURON® 3120™ CHIP and NEURON 3150™ CHIP

Advance Information
April 1992

ECHELON

TABLE OF CONTENTS

1 Introduction

1.1 General Description

The NEURON 3120 and 3150 CHIPS use CMOS VLSI technology to allow the implementation of low-cost local operating networks. Included in each NEURON CHIP are all the functions required to acquire and process information, make decisions, generate outputs and propagate control information, via a standard protocol, across a wide variety of network media such as twisted-pair, infrared, RF, powerline, or coaxial cable.

The chips are highly integrated and use a minimal number of external components. On chip are three 8-bit CPUs. One CPU executes the user's *application* such as measuring input parameters, timing events, making logical decisions, driving outputs, etc. The second CPU executes the *protocol*, properly encoding and decoding the messages to be sent over the network. The third CPU controls the *network communications* port which physically sends and receives the packets. To support these three CPUs, there is on-board EEPROM and RAM, and either on-board ROM (NEURON 3120 CHIP) or an external memory port (NEURON 3150 CHIP).

The chips can send and receive information on both the 5-pin communications port or via the 11-pin applications I/O port. The applications I/O port has 24 pre-programmed modes of operation to implement cost-effective measurement, timing and control applications.

1.2 Features

- Three 8-bit pipelined CPUs
  Selectable input clock rates: 625kHz to 10MHz
- On-chip memory
  2K-byte static RAM (NEURON 3150 CHIP)
  1K-byte static RAM (NEURON 3120 CHIP)
  512-byte EEPROM
  10K-byte ROM (NEURON 3120 CHIP)
- 11 programmable I/O pins
  24 selectable modes of operation
  Programmable pull-ups
  20 mA current sink
- Two 16-bit timer/counters for frequency and timer I/O
- Sleep mode for reduced current consumption while retaining operating state
- Network communications port
  Single-ended mode
  Differential mode
  Selectable transmission rates: 4.9 kbit/sec to 1.25 Mbit/sec
  280 packets/sec throughput sustained; 700 packets/sec peak
  40 mA current output for differentially driving twisted-pair networks
  Optional collision detect input
- Firmware
  LONTALK™ protocol conforming to 7-layer OSI reference model
  I/O drivers
  Event-driven task scheduler
- Service pin for remote identification and diagnostics
- Unique 48-bit internal NEURON ID

1.3 Applications

- Distributed sense and control systems
- Instrumentation
- Machine automation
- Process control
- Diagnostic equipment
- Environmental monitoring & control
- Power distribution & control
- Production control
- Lighting control
- Building automation & control
- Security systems
- Data collection/acquisition
- Robotics
- Home automation
- Consumer electronics
- Automotive electronics
- ....

2 LONWORKS™ Architectural Overview

LONWORKS tools and components form a complete platform for implementing intelligent distributed sense and control applications based on LONWORKS technology. These applications consist of intelligent nodes that interact with their environment, and communicate with one another over a variety of communications media using a common, message-based network protocol.

LONWORKS technology includes all of the elements required to design, deploy and support intelligent distributed control systems. Specifically, LONWORKS includes the following tools and components:

- NEURON CHIPs and associated firmware, including support for the LONTALK protocol
- LONBUILDER™ Developer's Workbench, a control network development system running on an IBM PC-compatible host computer, that enables easy development and debugging of LONWORKS nodes and integration of nodes into twisted-pair, powerline and RF networks.
- LONMANAGER™ network management tools

Each local point of control in a LONWORKS control network is called a node, and contains a NEURON CHIP, sense and control devices, a transceiver providing physical connection to the network medium, and a power source. See figure 2.1.

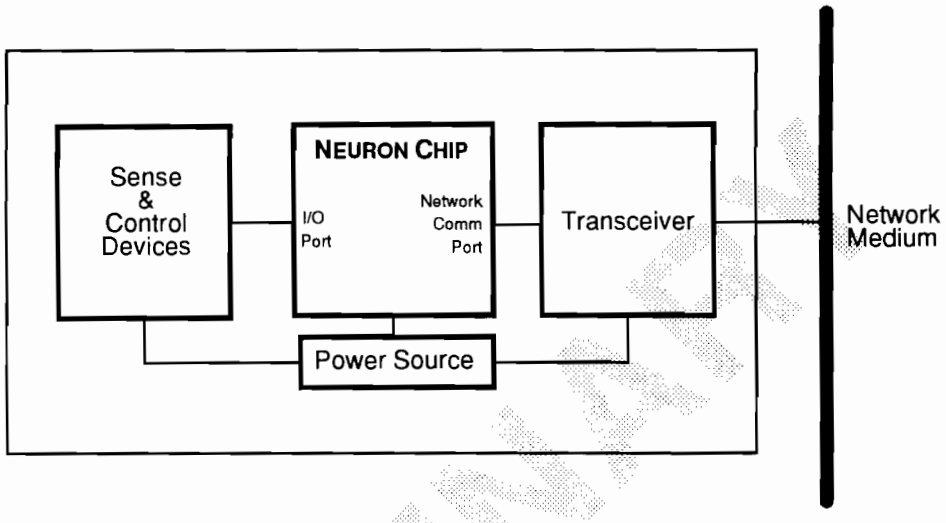


Figure 2.1 Block diagram of a typical LONWORKS node

The NEURON CHIP is the control and communications processor at the heart of the node. It is implemented as a single CMOS integrated circuit. Its key components include three 8-bit CPUs, on-board non-volatile and random-access memory, an applications I/O port and a network communications port which implements the LONTALK protocol.

The first members of the NEURON CHIP family are the NEURON 3120 and 3150 CHIPs. The NEURON 3120 CHIP has all of its memory on-board, while the NEURON 3150 CHIP has an external memory bus to support more complex applications. Please refer to table 2.1 and figures 2.2 and 2.3.

NEURON CHIP	3120	3150
CPUs	3	3
EEPROM bytes	512	512
RAM bytes	1,024	2,048
ROM bytes	10,240	0
External Memory Interface	No	Yes
16-bit Timer/Counters	2	2
Package	SOIC	PQFP
Pins	32	64

Table 2.1 Comparison of NEURON 3120 CHIP and NEURON 3150 CHIP

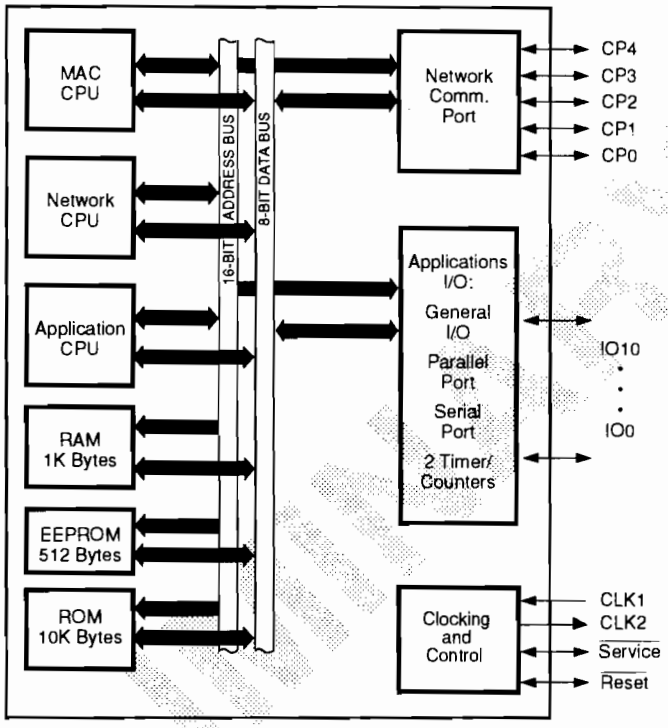


Figure 2.2 NEURON 3120 CHIP block diagram

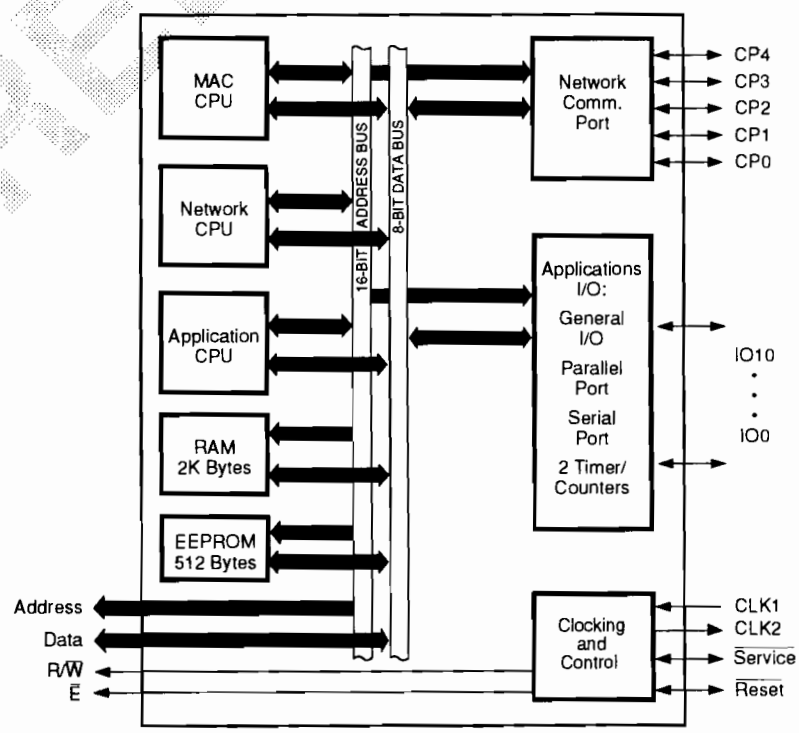


Figure 2.3 NEURON 3150 CHIP block diagram

3 CPUs

The NEURON CHIP contains three identical 8-bit central processing units (CPUs) which are dedicated to the following functions (see figure 3.1).

CPU-1 is the Media Access Control (MAC) Layer CPU that handles layer two of the seven-layer LONTALK protocol stack (layer one being the physical layer). CPU-1 processing includes driving the communications sub-system hardware as well as executing the media access algorithm. CPU-1 communicates with CPU-2 using network buffers located in shared memory (see figure 3.1).

CPU-2 is the Network CPU which implements layers three through six of the LONTALK protocol stack. It handles network variable processing, addressing, transaction processing, authentication, background diagnostics, software timers, network management, and routing functions if applicable. CPU-2 uses network buffers to communicate with CPU-1, and application buffers to communicate with CPU-3 (see figure 3.1). The buffers are also located in shared memory. Access to them is mediated with hardware semaphores to resolve contention when updating shared data.

CPU-3 is the Application CPU. It runs code written by the user, together with the operating system services called by applications code. The programming language used by the applications programmer is NEURON C, a derivative of the C language optimized and enhanced for LONWORKS distributed control applications. The major enhancements are the following:

- A built-in multi-tasking scheduler that allows the applications programmer to express logically parallel event-driven tasks in a natural way, and to control priority-based execution of these tasks.
- A declarative syntax for input/output objects directly mapping into the input/output capabilities of the NEURON CHIP (see section 8 for details).
- A declarative syntax for network variables, which are NEURON C language objects whose values are automatically propagated over the network whenever values are assigned to them.
- A declarative syntax for millisecond and second timer objects which activate user tasks on expiration.
- A library of functions, which when called, can perform event checking, manage input/output activities, send and receive messages across the network, and control miscellaneous functions of the NEURON CHIP.

The support for all these capabilities is part of the NEURON CHIP and does not need to be programmed by the user.

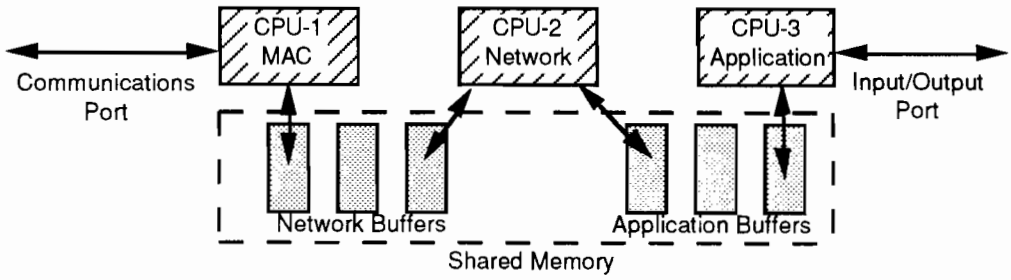


Figure 3.1 NEURON CHIP processor organization

Each of the three identical CPUs has its own register set (table 3.1), but all three CPUs share data and address ALUs and memory access circuitry (figure 3.2). Each CPU *minor* cycle consists of *three system clock* cycles, or phases. The minor cycles of the three CPUs are offset from one another by one system clock cycle, so that each CPU can access memory and ALUs once during each instruction cycle. Figure 3.2 shows the active elements for each CPU during one of the three phases of a minor cycle. The system thus pipelines the three processors, reducing hardware requirements without affecting performance. This allows the execution of three processes in parallel without time-consuming interrupts and context-switching.

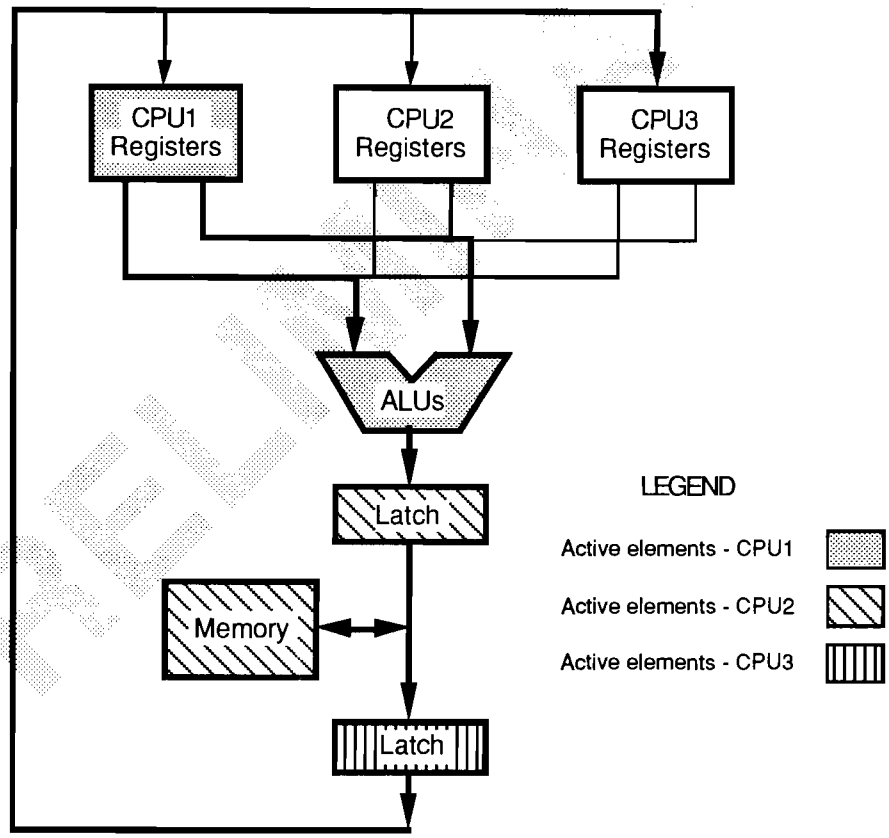


Figure 3.2 CPU/ memory activity during one of the three phases of a minor cycle

Mnemonic	Bits	Contents
FLAGS	8	CPU flags
IP	16	Next instruction pointer
BP	16	Address of 256-byte base page
DSP	8	Data stack pointer within base page
RSP	8	Return stack pointer within base page
TOS	8	Top of data stack

Table 3.1 CPU register set

The CPUs employ a stack architecture that results in very efficient memory utilization. Each CPU has two stacks. The *data* stack contains byte-wise operands. The *return* stack contains the return addresses from CALL instructions and may also be used for temporary data storage.

A CPU minor cycle is three system clock cycles, or six input clock cycles. Most instructions take between one and seven CPU minor cycles. At a maximum input clock rate of 10 MHz, instructions execute in 0.6 to 4.2 μseconds. The formula for instruction time is:

Instruction Time(μs) = # Cycles x 6 / Input Clock(MHz)

Tables 3.2, 3.3 and 3.4 list the CPU instructions and their cycle counts.

Mnemonic	# Cycles	Description
NOP	1	No operation
SBR	1	Short branch
BR/BRC/BRNC	2	Branch, branch on (not) carry
SBRZ/SBRNZ	3	Short branch on (not) zero
BRF	4	Branch far
BRZ/BRNZ	4	Branch on (not) zero
RET	4	Return from subroutine
BRNEQ	4/6	Branch if not equal (taken/not taken)
DBRNZ	5	Decrement and branch if not zero
CALLR	5	Call subroutine relative
CALL	6	Call subroutine
CALLF	7	Call subroutine far

Table 3.2 Program control instructions

Mnemonic	# Cycles	Addressing mode
PUSH cpureg	3	Any CPU register
POP cpureg	4	Any CPU register
PUSH/POP ID	4	Base page plus displacement (8-23)
PUSH/POP !TOS	4	Base page plus contents of TOS
PUSH [RSP]	4	Push data on top of return stack
PUSHS/PUSH #literal	4	Push 3 or 8-bit literal value
PUSHPOP	5	Pop from return stack, push to data stack
POPPUSH	5	Pop from data stack, push to return stack
LDBP address	5	Load base page pointer register
PUSH/POP [DSP][D]	5	Contents of DSP minus displacement (1-8)
PUSHD #literal	6	Push 16-bit literal value
PUSHD/POPD [PTR]	6	16-bit indirect through base page pointer (0-3)
PUSH/POP [PTR][TOS]	6	Indirect through base page pointer plus TOS
PUSH/POP [PTR][D]	7	Indirect through base page pointer plus displ.
PUSH/POP absolute	7	Push memory data to data stack
IN/OUT	7 + 4 n	Transfer *n* bytes to I/O

Table 3.3 Memory/stack instructions

Mnemonic	# Cycles	Operation
INC/DEC/NOT	2	Increment/decrement/negate TOS
ROLC/RORC	2	Rotate left/right TOS through carry
SHL/SHR	2	Logical left/right shift TOS
SHLA/SHRA	2	Arithmetic left/right shift TOS
DROP NEXT	2	Drop next element from data stack
DROP [RSP]	2	Drop top of return stack
ADD/AND/OR/XOR #literal	3	Operate with literal on TOS
DROP TOS	3	Drop top of data stack
DROP_R TOS	3	Drop top of data stack and return
ALLOC #literal	3	Move data stack pointer
ADD/AND/OR/XOR	4	Operate with next element on TOS

ADC/SBC/SUB/XCH	4	Operate with next element on TOS
DROP_R NEXT	5	Drop next element from data stack and return
INC [PTR]	6	Increment base page pointer (0-3)
DEALLOC #literal	6	Move data stack pointer and return
(ADD/AND/OR/XOR)_R	7	Operate with next element on TOS and return

Table 3.4 ALU instructions

4 Memory

The various components of NEURON CHIP memory are described below.

4.1 EEPROM

Both versions of the NEURON CHIP have 512 bytes of EEPROM containing:

- Network configuration and addressing information
- Unique 48-bit NEURON CHIP identification code
- 16 bits of manufacturer's code
- User-written application code and read-mostly data

All but eight bytes of the EEPROM can be written under program control. The NEURON CHIP uses an on-board charge pump to generate the required programming voltage. The charge pump operation is transparent to the user. The remaining eight bytes are permanently written during chip manufacture, and include a unique 48-bit identifier (NEURON ID) for each part. The total erase and write time is 20 ms per byte. EEPROM may be written 10,000 times with no data loss. For both the NEURON 3120 CHIP and the NEURON 3150 CHIP, the EEPROM stores installation-specific information such as network addresses and communications parameters. For the NEURON 3120 CHIP, EEPROM memory also stores the application program generated by the LONBUILDER Developer's Workbench. The application code for the NEURON 3150 CHIP may be stored either on-chip in the EEPROM memory, or off-chip in external memory, or both.

Note that when the NEURON CHIP is not within the specified power supply voltage range, a pending or on-going EEPROM write is not guaranteed. While there is built-in protection to prevent EEPROM corruption during power-down, it is important to hold the ~RESET pin low whenever V_{DD} is below its recommended operating condition to avoid this possibility.

4.2 RAM

The NEURON 3120 CHIP contains 1024 bytes of on-chip RAM (random-access memory).

The NEURON 3150 CHIP contains 2048 bytes of on-chip RAM. The RAM is used to store:

- Stack segment, application, and system data
- LONTALK protocol network buffers and application buffers

The RAM state is retained as long as power is applied to the device, even in "sleep" mode.

4.3 ROM

The NEURON 3120 CHIP includes 10,240 bytes of ROM (read-only memory). This memory stores the NEURON CHIP firmware, including:

- LONTALK protocol code

- Event-driven task scheduler
- Application function libraries

4.4 External Memory

The NEURON 3150 CHIP does not include any on-chip ROM. Instead, it allows addressing of up to 59,392 bytes of external memory. External memory is used for:

- Application program and data (up to 43,008 bytes)
- NEURON CHIP firmware & reserved space (16,384 bytes)
- LONTALK protocol code
- Event-driven task scheduler
- Application function libraries

The 43,008 bytes of space available for application program and data may also contain additional network buffers and application buffers.

The external memory space can be populated with combinations of RAM, ROM, PROM, EPROM or EEPROM in 256 byte increments. The memory maps are shown in figures 4.1 and 4.2.

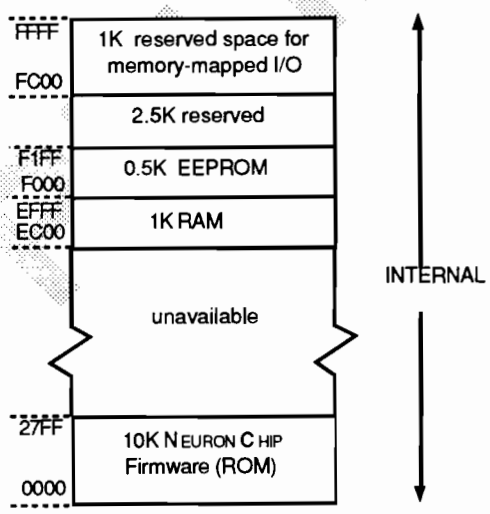


Figure 4.1 NEURON 3120 CHIP Memory map

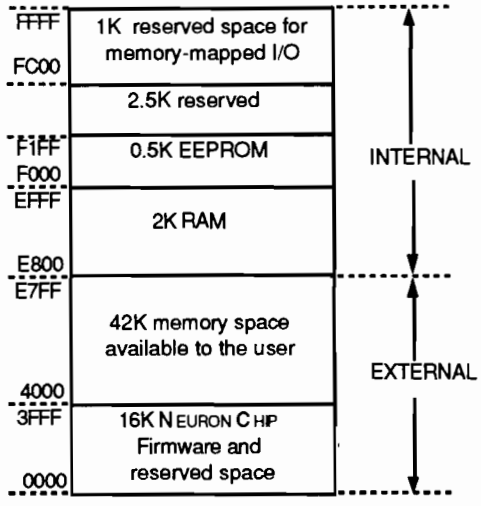


Figure 4.2 NEURON 3150 CHIP Memory map

The external memory bus has eight bidirectional data lines, sixteen address lines and two control outputs driven by the NEURON CHIP. At maximum clock rate (10 MHz input clock), 70 ns or better access time to external memory is required. If the input clock is scaled down, slower memory can be used.

Pin Designation	Direction	Function
A0-A15	Output	Address pins
D0-D7	Input/Output	Data pins
~E	Output	Enable clock
R/~W	Output	Read/~Write Select

Table 4.1 NEURON 3150 CHIP external memory interface pins

The Enable Clock (~E) runs at the system clock rate, which is one half the input clock rate. The Enable Clock is low whenever data is being transferred between the NEURON CHIP and external memory. All memory, both internal and external, may be accessed by any of the three CPUs at the appropriate phase of the instruction cycle. Since the instruction cycles of the three CPUs are offset by one system clock cycle with respect to each other, the memory bus is in use by only one CPU at a time.

Figure 4.3 below illustrates an example of the NEURON CHIP connected to a read-only-type memory. In this example, note that when A15 is low, the NEURON CHIP accesses external memory at addresses 0000 to 7FFF. Pins ~E and R/~W are not used. The LONWORKS Engineering Bulletin *The NEURON 3150 CHIP External Memory Interface* shows how to connect other types of memory. For memory interface timing, see the electrical specifications section.

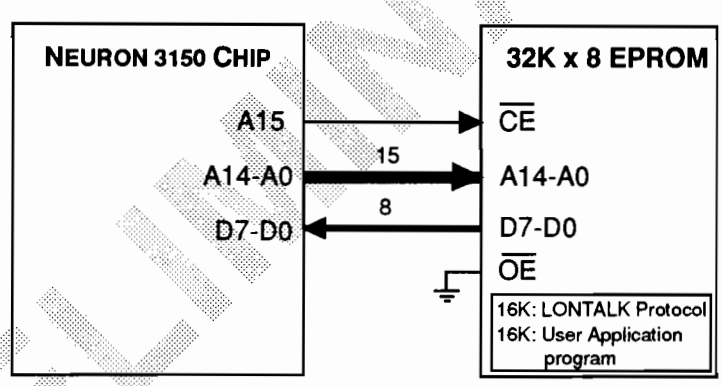


Figure. 4.3 Connecting the NEURON 3150 CHIP to an EPROM

5 LONTALK Protocol

The NEURON CHIP implements a complete networking protocol using two of the three on-chip CPUs. This networking protocol follows the ISO OSI reference model for network protocols; this supports flexible addressing and multiple communications channels on a single network. Application programs running on NEURON CHIP nodes use this protocol to communicate with applications running on other NEURON CHIP nodes elsewhere on the same network. See sections 7.2 and 7.3 for details of how the protocol is used by application-level objects called network variables and message tags. The three processors in the NEURON CHIP are used to execute the protocol software along with the application program. Table 5.1 shows the mapping of LONTALK services onto the 7-layer OSI reference model.

OSI Layer		Purpose	Services Provided	Processor
7	Application	Application compatibility	Standard Network Variable Types	APP
6	Presentation	Data interpretation	Network variables, foreign frame transmission	NET
5	Session	Remote actions	Request/response, authentication, network management	NET
4	Transport	End-to-end reliability	Acknowledged and unacknowledged, unicast and multicast, authentication, common ordering , duplicate detection	NET
3	Network	Destination addressing	Addressing, routers	NET

2	Link	Media access and framing	Framing, data encoding, CRC error checking, predictive CSMA, collision avoidance, priority, collision detection	MAC
1	Physical	Electrical interconnect	Media-specific interfaces and modulation schemes	MAC, XCVR

Table 5.1 LONTALK protocol layering

The main features of the LONTALK protocol are the following:

5.1 Multiple Media Support

The protocol processing on the NEURON CHIP is media-independent. This allows the NEURON CHIP to support a wide variety of communications media, including twisted-pair, powerline, radio-frequency, infrared, coaxial cable and fiber optics.

5.2 Support for Multiple Communication Channels

A channel is a physical transport medium for packets. A network may be composed of one or more channels. In order for packets to be transferred from one channel to another, a device called a router is used to connect the two channels. These devices typically consist of two NEURON CHIPs connected together via their I/O pins. Each of the NEURON CHIPs uses its own transceiver to communicate with its channel. The protocol supports such a device so that multi-media networks may be constructed, and network loading can be optimized by localizing traffic.

5.3 Communications Rates

Channels may be configured for different bit rates to allow trade-offs of distance, throughput, and power consumption. The allowable bit rates are 4.9, 9.8, 19.5, 39.1, 78.1, 156.3, 312.5, 625 and 1,250 kbit/s. Channel throughput depends on the bit rate, the average size of a packet, and the use of acknowledgments, priority and authentication. An average packet is in the range of 10 to 16 bytes long, depending on the length of the domain identifier, the addressing mode, size of the data field for a network variable update or an explicit message.

5.4 LONTALK Addressing Limits

The top level of the addressing hierarchy is a domain. For example, if different network applications are implemented on a shared communications medium such as RF, different domain identifiers can be used to keep the applications completely separate. The domain identifier is selectable to be 0, 1, 3 or 6 bytes long.

The second level of addressing is the subnet. There may be up to 255 subnets per domain. A subnet is a logical grouping of nodes from one or more channels. An intelligent router operates at the subnet level. It determines which subnets lie on which side of it, and forwards packets accordingly.

The third level of addressing is the node. There may be up to 127 nodes per subnet. Thus a maximum of 255 x 127 = 32,385 nodes may be in a single domain. Any node may be a member of one or two domains, allowing a node to serve as an inter-domain gateway. This also allows, for example, a single sensor node to transmit its outputs into two different domains.

Nodes may also be grouped. Groups of nodes may span several subnets within a domain. Groups may also span several channels. Up to 256 groups may be specified within a domain, and up to 64 nodes may be in a group for acknowledged service. For

unacknowledged service within a domain, an unlimited number of nodes may belong to a group. A single node may be a member of up to 15 groups. Group addressing reduces the number of bytes of address information transmitted with each message, and also allows many nodes to receive a piece of information using a single message on the network.

In addition, each node carries a unique 48-bit NEURON ID assigned during manufacture. This ID is typically used as a network address only during installation and configuration. It may also be read and used by application programs as a unique product serial number.

The channel (transmission medium) of a node does not affect the way a node is addressed. Domains can contain several channels. Subnets and groups may also span several channels.

Nodes are addressed using one of five addressing formats:

Address Data Specified		Nodes Addressed
Domain,	Subnet = 0	All nodes in the domain
Domain,	Subnet	All nodes in the subnet
Domain,	Subnet, Node	Specific logical node
Domain,	Group	All nodes in the group
NEURON ID		Specific physical node

5.5 Message Services

The LONTALK protocol offers four basic types of message service. The first two service types are end-to-end acknowledged. They are the following:

ACKD, or end-to-end acknowledged service, where a message is sent to a node or group of nodes, and individual acknowledgments are expected from each receiver. If the acknowledgments are not all received, the sender times out and re-tries the transaction. The number of re-tries and the time-out are both selectable. The acknowledgments are generated by the network CPU without intervention of the application. Transaction IDs keep track of messages and acknowledgments so that an application does not receive duplicate messages.

REQUEST, or request/response service, where a message is sent to a node or group of nodes, and individual responses are expected from each receiver. The incoming message is processed by the application on the receiving side before a response is generated. The same retry and time-out options are available as with ACKD service. Responses may include data, so that this service is particularly suitable for remote procedure call, or client/server applications.

The second two service types are unacknowledged. They are the following:

UNACKD_RPT (unacknowledged repeated), where a message is sent to a node or group of nodes multiple times, and no response is expected. This service is typically used when multicasting to large groups of nodes, in which the traffic generated by all the responses would overload the network.

UNACKD (unacknowledged), where a message is sent once to a node or group of nodes, and no response is expected. This is typically used when highest transmission rate is required, or when large amounts of data are to be transferred. When using this service, the application must not be sensitive to the loss of a message.

5.6 Authentication

The LONTALK protocol supports authenticated messages, which allow the receivers of a message to determine if the sender is authorized to send that message. This is used to prevent unauthorized access to nodes and their applications. Authentication is implemented by distributing 48-bit keys to the nodes at installation time. For an

authenticated message to be accepted by the receiver, both sender and receiver must possess the same key. This key is distinct from the node's NEURON ID.

When an authenticated message is sent, the receiver challenges the sender to provide authentication, using a different random challenge every time. The sender then responds with a transformation performed on the challenge, using the authentication key. The receiver compares the reply to the challenge with its own transformation on the challenge. If the transformations match, the transaction goes forward. The transformation used is designed so that it is extremely difficult to deduce what the key is, even if the challenge and the response are both known. The use of authentication is configurable individually for each network variable connection. In addition, network management transactions may be optionally authenticated.

5.7 Priority

The LONTALK protocol optionally offers a priority mechanism to improve the response time of critical packets. The protocol permits the user to specify priority time slots on a channel, dedicated to priority nodes. Each priority time slot on a channel adds time to the transmission of every message, but now dedicated bandwidth is available at the end of each packet for priority access without any contention for the channel

5.8 Collision Avoidance

The LONTALK protocol uses a unique collision avoidance algorithm which has the property that under conditions of overload, the channel can still carry its maximum capacity, rather than have its throughput degrade due to excess collisions.

5.9 Collision Detection

On communications media that support hardware collision detection (for example, twisted pair), the LONTALK protocol can optionally cancel transmission of a packet as soon as a collision is detected by the transceiver. This allows the node to immediately retransmit any packet that has been damaged by a collision. Retransmission after a collision has been detected occurs for all classes of service. Without collision detection, the node would have to wait for the retry time before retransmitting the packet, assuming acknowledged or request/response service.

5.10 Foreign Frames

A special range of message codes is reserved for foreign frame transmission. Up to 229 bytes of data may be embedded in a message packet and transmitted like any other message. The LONTALK protocol applies no special processing to foreign frames, they are treated as a simple array of bytes. The application program may interpret the data in any way it wishes.

5.11 Network Management and Diagnostic Services

Refer to Appendix B.

6 Network Communications Port

There are three modes of operation for the NEURON CHIP's 5-pin communications port. They are single-ended, differential and special-purpose. Table 6.1 summarizes the pin assignments for each mode.

Pin	Single-Ended Mode	Differential Mode	Special-Purpose Mode
CP0	Data input	+ Data input	RX input
CP1	Data output	- Data input	TX output
CP2	Transmit Enable output	+ Data output	Bit clock output
CP3	~Sleep (~Power-down) output	- Data output	~Sleep output or Wake-up input
CP4	~Collision Detect input	~Collision Detect input	Frame clock output

Table 6.1 Network communications port pin assignments

Single-ended and differential modes use Differential Manchester encoding which is a widely used and reliable format for transmitting data over various media. Differential Manchester coding is polarity-insensitive. Thus, reversal of polarity in the communication link will not affect data reception. The available network bit rates for single-ended and differential modes are given in table 6.2, as a function of the NEURON CHIP's input clock rate.

Network Bit Rate (kbit/sec)	Minimum Input Clock (MHz)	Maximum Input Clock (MHz)
1,250	10.0	10.0
625	5.0	10.0
312.5	2.5	10.0
156.3	1.25	10.0
78.1	0.625	10.0
39.1	0.625	10.0
19.5	0.625	10.0
9.8	0.625	10.0
4.9	0.625	5.0

Table 6.2 Single-ended and differential network data rates

6.1 Single-Ended Mode

The single-ended mode is most commonly used with external active transceivers interfacing to media such as RF , IR, fiber optic, and coaxial cable.

Figure 6.1 shows the communications port configuration for the single-ended mode of operation. Data communication occurs via the single-ended (with respect to V_{SS}) input and output buffers on pins CP0 and CP1.

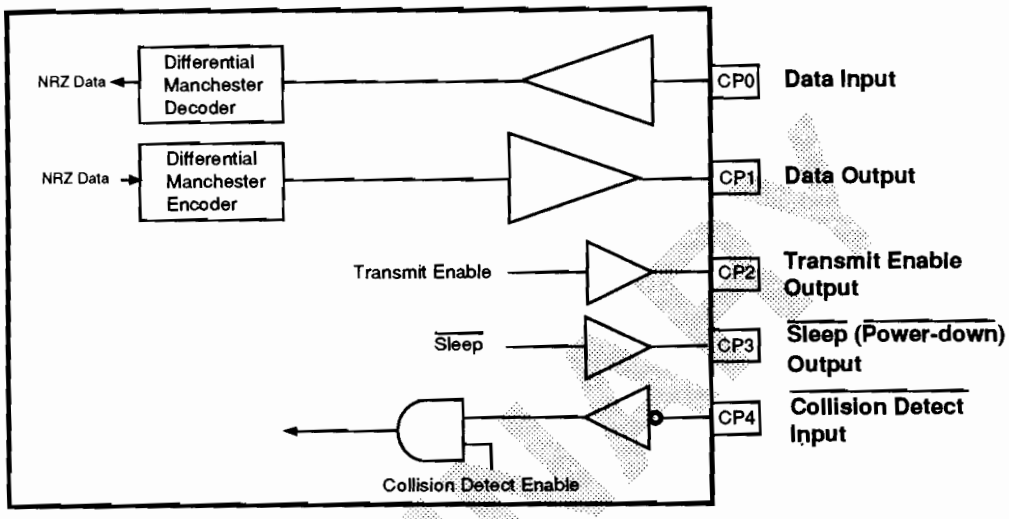


Figure 6.1 Single-ended communications mode configuration

In single-ended mode, the communications port encodes transmitted data and decodes received data using Differential Manchester coding (also known as bi-phase space coding). This scheme guarantees a transition at the beginning of every bit period for the purpose of synchronizing the receiver clock. Zero/one data is indicated by the presence or absence of a second transition halfway between clock transitions. A mid-cell transition indicates a 0. Lack of a mid-cell transition indicates a 1. Figure 6.2 below shows a typical packet where T is the bit period, equal to 1/(bit rate).

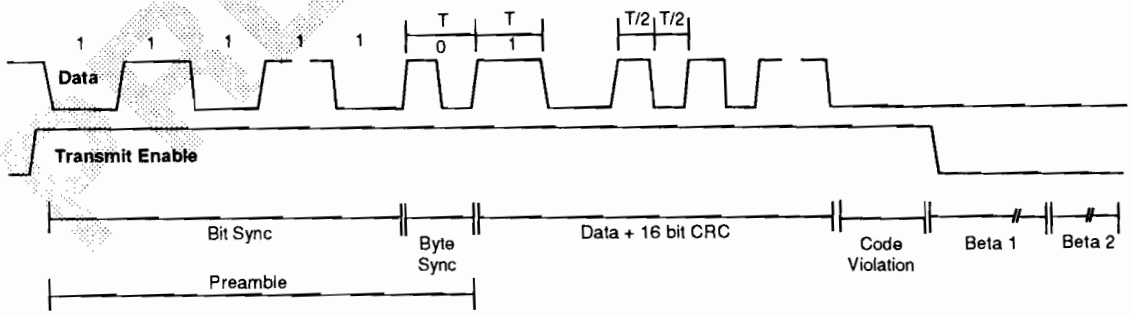


Figure 6.2 Single-ended mode data format

The transmitter transmits a *preamble* at the beginning of a packet to allow the other nodes to synchronize their receiver clocks. The preamble consists of a series of Differential Manchester 1's. Its duration is at least six bits long and is selectable by the user.

The preamble ends with a single byte-sync bit, which marks the start of byte boundaries at the bit following. The byte-sync bit is a Differential Manchester 0.

The NEURON CHIP terminates the packet by forcing a Differential Manchester code violation. After sending the last CRC bit, it holds the data output transitionless for 2-1/2 bit times. The Transmit Enable pin on CP2 is then driven low, indicating the end of transmission. For a NEURON CHIP, the time to format and begin to send a 12-byte message is from 2.8 to 44.8 ms depending on the system clock rate (10MHz to 625KHz).

As an option, the NEURON CHIP accepts an active-low Collision Detect input from the transceiver. If collision detection is enabled and CP4 goes low for at least one system clock period (200 ns with a 10 MHz clock) during transmission, the NEURON CHIP is signaled that a collision has or is occurring and that the message must be resent. The node then attempts to reaccess the channel.

If the node does not use collision detection, then the only way it can determine that a message has not been received is to request an acknowledgement. When acknowledged service is used, the retry timer is set to allow sufficient time for a message to be sent and acknowledged (typically 48 to 96 ms at 1.25 Mbps when there are no routers in the transmission path). If the retry timer times out, the node attempts to reaccess the channel. The benefit of using collision detection is that the node does not have to wait for the retry timer to time-out before attempting to resend the message, because the node detects the collision when it sends the packet.

Beta 1 time is the idle period after a packet has been sent.

Both priority (P) and non-priority slots are defined by the Beta 2 time. Nodes listen to the network prior to transmitting a packet. This prevents nodes from transmitting packets on top of each other except when the packets are initiated at nearly the same time. In addition, nodes randomize the time before they start transmitting on the network. When the network is idle, all nodes randomize over 16 slots. When the estimated network load increases, nodes start randomizing over more slots in order to lower the probability of a collision. The number of randomizing slots (R) varies from 16 up to 1008, based on "n", the estimated channel backlog (the number of slots is n*16 where "n" has a range of 1 to 63).

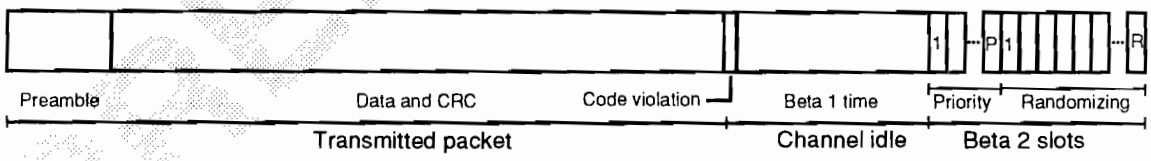


Figure 6.3 Packet timing

In order for the receiver to detect the edge transitions, two windows are set up for each bit period, T. The first window is set at T/2 and determines if a 0 is being received. The second window is at T and defines a 1. This transition then sets up the next two windows (T/2 and T). If no transition occurs, a Manchester code violation is detected and the packet is assumed to have ended. Table 6.3 shows the width of this window as a function of the ratio of the NEURON CHIP input clock (MHz) and the network bit rate (Mbps) selected. If a transition falls outside of either window, it is not detected. Timing instability of the transitions, known as jitter, may be caused by changes in the communications medium, or instability in the transmitting or receiving nodes' input clocks. The jitter tolerance windows are expressed as fractions of the bit period, T, in table 6.3.

Ratio of NEURON CHIP Input Clock to Network Bit Rate	Next Data Edge			Next Clock Edge		
	Minimum	Nominal	Maximum	Minimum	Nominal	Maximum
8:1	0.375T	0.500T	0.622T	0.875T	1.000T	1.122T
16:1	0.313T	0.500T	0.685T	0.813T	1.000T	1.185T
32:1	0.345T	0.500T	0.717T	0.845T	1.000T	1.155T
64:1	0.330T	0.500T	0.702T	0.830T	1.000T	1.170T
128:1	0.323T	0.500T	0.695T	0.823T	1.000T	1.177T
256:1	0.313T	0.500T	0.690T	0.818T	1.000T	1.182T
512:1	0.315T	0.500T	0.687T	0.815T	1.000T	1.185T
1024:1	0.315T	0.500T	0.687T	0.815T	1.000T	1.185T

Table 6.3 Receiver jitter tolerance windows

For further information, refer to Echelon's two Engineering Bulletins: *Enhanced Media Access Control with Echelon's LONTALK Protocol* and *Implementing Twisted Pair Transceivers with the NEURON CHIP.*

6.2 Differential Mode

In differential mode, the NEURON CHIP's built-in transceiver is able to differentially drive and sense a twisted-pair transmission line with external passive components. Differential mode is similar in most respects to single-ended mode; the key difference is that the driver/receiver circuitry is configured for differential line transmission. Data output pins CP2 and CP3 are driven to opposite states during transmission and put in a high-impedance (undriven state) when not transmitting. The differential receiver circuitry on pins CP0 and CP1 has selectable hysteresis with eight selectable voltage levels followed by a selectable low-pass filter with four selectable values of transient pulse (noise) suppression. The selectable hysteresis and filter permit optimizing receiver performance to line conditions. See the Electrical Specification (tables 12.5 and 12.6) for specific values.

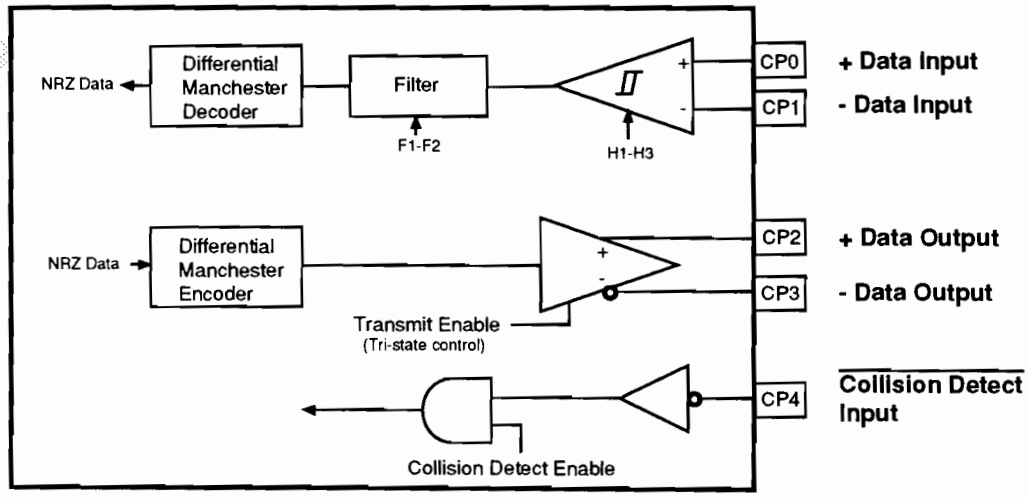


Figure 6.4 Differential communications mode

Figure 6.5 shows a typical packet waveform in differential mode. Note that the packet format is identical to that of single-ended described earlier; the coding and jitter tolerance also apply identically.

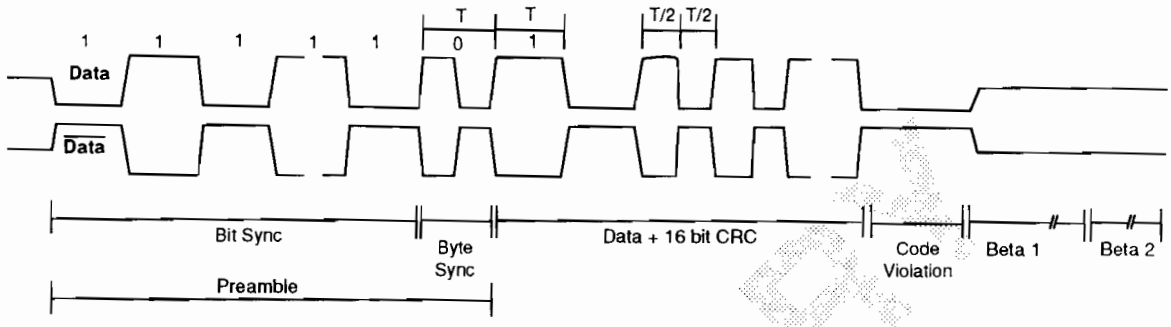


Figure 6.5 Differential mode data format

6.3 Special-Purpose Mode

In special situations, it is desirable for the NEURON CHIP to provide the packet data in an unencoded format and without a preamble. In this case, an intelligent transmitter accepts the unencoded data and does its own formatting and preamble insertion. The intelligent receiver then detects and strips off the preamble and formatting and returns the decoded data to the NEURON CHIP.

Such an intelligent transceiver contains its own input and output data buffers, intelligent control functions, and provides handshaking signals to properly pass the data back and forth between the NEURON CHIP and the transceiver.

In addition, there are many features that can be defined by and incorporated into a special purpose transceiver.

- Ability to configure various parameters of the transceiver from the NEURON CHIP
- Ability to report on various parameters of the transceiver to the NEURON CHIP
- Multiple channel operation
- Multiple bit rate operation
- Use of forward error correction
- Media specific modulation techniques requiring special message headers and framing

When the special-purpose mode is used, a protocol is utilized between the NEURON CHIP and the transceiver. This protocol consists of the NEURON CHIP and the transceiver, each exchanging 16 bits, 8 bits of status and 8 bits of data, (see figure 6.6) simultaneously and continuously at rates up to 1.25 Mbps (when the NEURON CHIP's input clock is 10 MHz). The 1.25 Mbps bit rate allows time-critical flags, such as a Carrier Detect, to be exchanged across the interface with network bit rates up to 156 Kbps. The maximum network bit rate is 156Kbps due to the overhead associated with the handshaking.

There are three types of data that can be sent and received during each frame.

1. Network packet data: Actual data (8 bits at a time) that is to be transmitted or received.

2. Configuration data: Information from the NEURON CHIP which tells the transceiver how it is to be set up or configured.

3. Status data: Information parameters reported from the transceiver to the NEURON CHIP, when it is requested by the NEURON CHIP.

The contents of the configuration data and status data are defined by the transceiver. The status flags are listed in table 6.4.

If it is determined that the special-purpose mode is necessary and practical for the user's transceiver design then the LONBUILDER Developer's Workbench can be used to test this mode of operation with the user's own transceiver.

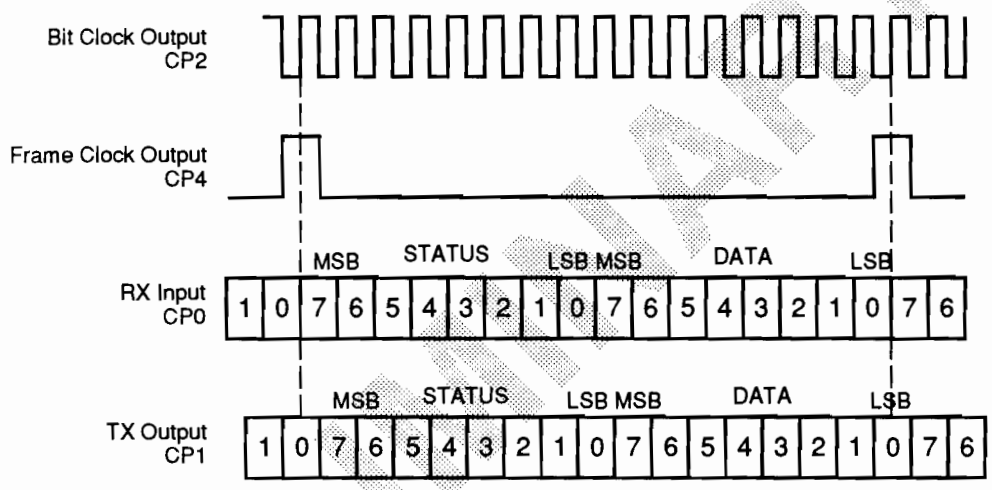


Figure 6.6 Special-purpose mode data format

TX Output, Status bits

bit 7	TX_FLAG	NEURON CHIP in the process of transmitting packet
bit 6	TX_REQ_FLAG	NEURON CHIP requests to transmit on the network
bit 5	TX_DATA_VALID	NEURON CHIP is passing network data to the transceiver in this frame
bit 4	Don't Care	Unused
bit 3	TX_ADDR_R/W	If negated, NEURON CHIP is writing internal transceiver register
bit 2	TX_ADDR_2	Address bit 2 of internal transceiver register (1 .. 7) *
bit1	TX_ADDR_1	Address bit 1 of internal transceiver register (1 .. 7) *
bit 0	TX_ADDR_0	Address bit 0 of internal transceiver register (1 .. 7) *

RX Input, Status bits

bit 7	SET_TX_FLAG	Transceiver accepts request to transmit packet
bit 6	CLR_TX_REQ_FLAG	Transceiver acknowledges request to transmit packet
bit 5	RX_DATA_VALID	Transceiver is passing network data to the NEURON CHIP in this frame
bit 4	TX_DATA_CTS	Transceiver indicates that NEURON CHIP is clear to send byte of network data
bit 3	SET_COLL_DET	Transceiver has detected a collision while transmitting the preamble
bit 2	RX_FLAG	Transceiver has detected a packet on the network
bit 1	RD/WR_ACK	Transceiver acknowledges read/write to internal register
bit 0	TX_ON	Transceiver is transmitting on the network

* Note that internal transceiver Register 0 is not valid. Registers 1 through 7 are defined by the transceiver implementation.

Table 6.4 Special-purpose mode transmit and receive status bits

While the special purpose mode offers custom features, it is expected that most transceivers will use the single-ended mode for most types of medium, from coaxial

cable to RF to fiber optic, since it offers Differential Manchester encoding, which takes care of clock recovery.

7 Programming Model

The programming language used to write applications for the NEURON CHIP is a derivative of the C programming language called NEURON C. NEURON C is based on ANSI C, enhanced to support I/O, event processing, message passing, and distributed data objects. Several major differences exist between NEURON C and ANSI C in the data types supported. The numeric data types supported by NEURON C are the following:

`char`	8 bits	signed or unsigned
`short`	8 bits	signed or unsigned
`int`	8 bits	signed or unsigned
`long`	16 bits	signed or unsigned

NEURON C also supports *typedefs*, *enums*, arrays, pointers, *structs* and *unions*. Unlike ANSI C, NEURON C does not include a standard run-time library supporting file I/O and other features common to larger target processors, such as floating point. However, NEURON C has a special run-time library and language syntax extensions supporting intelligent distributed control applications using NEURON CHIPs. NEURON C extensions include software timers, network variables, explicit messages, a multi-tasking scheduler, EEPROM variables and miscellaneous functions. A function library implementing floating point, 32-bit fixed point and string operations is also available for NEURON C. For further details on the NEURON C programming language, see the *NEURON C Programmer's Guide.*

7.1 Timers

An application program can use up to fifteen timers that decrement either every second or every millisecond and optionally repeat. These timers are implemented in software running on the Network CPU, and are independent of the NEURON CHIP input clock rate. The expiration of a timer is an event that may cause the execution of a user-written task (see below). This event is called *timer_expires.* The value of a timer variable is an unsigned long (0-65,535).

7.2 Network Variables

The application program can declare a special class of static objects called network variables, which may be of class *input* or *output.* Assignment of a value to a output network variable causes propagation of that value to all nodes declaring an input variable that is connected to the output network variable. For example, a node that contains a temperature sensing device could declare an output network variable which contains the current temperature sensed by the node. Every time the node measures a new value for the temperature, it updates the output network variable. Another node or nodes needing to know the current temperature, such as a heating control node, can then declare an input network variable for current temperature. Whenever the heating control node wants to use the value of the current temperature, it simply refers to the input network variable which will always contain the last temperature measured by the sensing node. At installation time, the output network variable on the sensing node is connected to the input network variable on the controlling node.

Binding is the process of connecting network variables from different nodes together, and is typically performed by a network management tool. The LONBUILDER Developer's Workbench and the LONMANAGER™ Application Programming Interface (API) both include such a binding capability. The binding is physically implemented by sending

network management messages containing the necessary addressing information to the nodes to be bound. Nodes may also update their own binding information for simple networks, without network management tools. Tables containing binding information are in the EEPROM, and hence may be written after the node has been manufactured.

Nodes declaring a input network variable need only refer to that variable to determine the latest value propagated across the network. A node declaring an input network variable may also call the library routine *poll* to cause the latest value to be propagated to it. The library routine *is_bound* may be used to check if a network variable is associated with (bound to) a network variable on another node.

Note that declaring network variables within a node's code occurs at compile time. The binding of the network variable outputs from one node to inputs on other nodes occurs at a later time, either before, during or after the node is installed in a network.

The network variable concept greatly simplifies the programming of complex distributed applications. Network variables provide a very flexible view of distributed data to be operated on by the nodes in a system. The programmer need not deal with message buffers, node addressing, request/response/retry processing and other low-level details.

A node running a regular application program may have up to 62 network variables, including array elements. A network variable may be a NEURON C variable or structure up to 31 bytes in length. Arrays of up to 31 bytes may be embedded in a structure and propagated as a single network variable. A network variable may also be an array of elements, each of which may be individually bound to network variables on other nodes. An output network variable on a node may also be connected to an input network variable on the same node (turnaround network variable).

Nodes can be implemented using the NEURON CHIP as a communications processor and a second processor as the host. The NEURON CHIP runs the LONWORKS Microprocessor Interface Program (MIP) as its application program. The MIP is a special application that can be used to move network variable processing off the NEURON CHIP and onto the host processor. In this case, the network variable information is located in the host's memory. Using the MIP, nodes may have up to 4096 network variables. The actual limit may be less depending on the network management tool used for binding the network variables; for example, the LONMANAGER API release 2.0 is limited to 255 network variables per node.

Network variable updates may be sent with four classes of service:

ACKD	Acknowledged service with retries
UNACKD	Unacknowledged service
UNACKD_RPT	Unacknowledged repeated service (message sent multiple times)
REQUEST	Request/response service, used for polling network variables

Network variables may be specified as authenticated, meaning that authenticated messages are used to transmit their values (see section 5.6). Network variables may also be specified to have priority, meaning that a priority time slot is used to transmit their values (see section 5.7).

The following built-in events may be checked by the scheduler to allow asynchronous processing of network variables:

nv_update_occurs	A new value has been received for an input network variable
nv_update_fails	Propagation of the value of an output network variable has failed
nv_update_succeeds	Propagation of the value of an output network variable has succeeded

nv_update_completes Propagation of the value of an output network variable has completed, either successfully or unsuccessfully

7.3 Explicit Messages

For most applications, network variables allow the most compact and simple implementation. However, for applications where data objects larger than 31 bytes need to be transmitted, request/response service is desired, or the network variable model is not suitable, then the application can use explicit messaging.

The application program can construct messages containing up to 229 bytes of data, addressed to other nodes or groups of nodes via implicit address connections called message tags. Alternatively, messages may be explicitly addressed to other nodes using subnet/node, group, broadcast or NEURON ID addressing. The messages may be sent with four classes of service:

ACKD Acknowledged service with retry

UNACKD Unacknowledged service

UNACKD_RPT Unacknowledged repeated service (message sent multiple times)

REQUEST Request/response service

Request/response service allows the node receiving a message to respond with data in the response message, as distinct from an acknowledgment, which contains no application-level data. Messages may be specified as authenticated (see section 5.6). Messages may also be specified to have priority, meaning that they use a priority time slot on the channel if the node has a priority slot allocated to it. In any case, priority messages are always sent by the node before non-priority messages (see section 5.7). The application program may explicitly assign the destination addresses or use the default address associated with the message tag.

Messages are exchanged between nodes by calling run-time library support routines:

msg_alloc	Allocate a message buffer
msg_alloc_priority	Allocate a priority message buffer
msg_cancel	Cancel a message being built for sending
msg_free	Free a message buffer
msg_receive	Receive a message at this node
msg_send	Send a message to another node
resp_alloc	Allocate a buffer for a response to a request message
resp_cancel	Cancel a response being built
resp_free	Free a response buffer
resp_receive	Receive a response to a request message
resp_send	Send a response to a request message

The following built-in events may be checked by the scheduler to allow asynchronous transmission and reception of messages:

msg_arrives	A message has arrived at this node
msg_completes	An outgoing message has been handled, either successfully or unsuccessfully
msg_succeeds	An outgoing message has been successfully sent
msg_fails	Some acknowledgements have not been received for an outgoing message
resp_arrives	A response to a request has been received

The following data objects are defined for use by the application program:

msg_in	Currently received message
msg_out	Message to be sent
resp_in	Currently received response to a previous message

7.4 Scheduler

The scheduler executes user-written tasks in response to events or conditions specified in *when* clauses by the application program. When a specified event or condition becomes true, the associated task body is executed. The user has the capability of specifying one or more *when* clauses as having priority, and the scheduler checks priority *when* clauses, one per scheduler cycle, in order of their appearance in the NEURON C program, followed by the non-priority *when* clauses in a round-robin fashion. The task scheduler can handle up to eighty *when* clauses, depending on type.

Events fall into five classes:

System wide events

reset	Node has been reset
offline	Node has been set off-line
online	Node has been set on-line
flush_completes	Node has completed preparations to enter sleep mode
wink	Node has received 'wink' network management message

Input/Output Events

io_changes	Value read from an I/O object has changed since last reading. Changes may be unconditional, or changed *to* a specified value, or *by* a specified amount.
io_update_occurs	Value read from a timer/counter input object has been updated
io_in_ready	Parallel I/O object is ready to receive data from external CPU
io_out_ready	Parallel I/O object is ready to transmit data to external CPU

Timer Events

timer_expires	Software timer value has decremented to zero

Message and Network Variable Events

The following events associated with the transmission of network variables and messages are discussed above in sections 7.2 and 7.3:

nv_update_occurs, nv_update_fails, nv_update_succeeds, nv_update_completes
msg_arrives, msg_completes, msg_succeeds, msg_fails, resp_arrives

User-Specified Events

<boolean expression>	User-specified expression, evaluating to true or false

7.5 Additional Library Functions

The following miscellaneous functions are available from the application program:

Execution Control:

delay	Delay processing for a time independent of input clock rate
flush_cancel	Cancel a flush in progress
flush_wait	Wait for outgoing messages and updates to be sent before going offline
go_offline	Cease execution of the application program
post_events	Defines a critical section for network variable and message processing
power_up	Determine whether last processor reset was due to power up
scaled_delay	Delay processing for a time that depends on the input clock rate
watchdog_update	Tickle the watchdog timer to prevent node reset

Network Management/Control:

access_address	Read node's address table
access_domain	Read node's domain table
access_nv	Read node's network variable configuration table
go_unconfigured	Reset this node to an uninstalled state
offline_confirm	Inform network management node that this node is going off-line
update_address	Write node's address table
update_domain	Write node's domain table
update_nv	Write node's network variable configuration table

Error Handling:

application_restart	Begin application program over again
clear_status	Clear error statistics accumulators and error log
error_log	Record software-detected error
node_reset	Activate NEURON CHIP reset pin, and reset all CPUs
retrieve_status	Read error statistics from protocol processor
retrieve_xcvr_status	Read transceiver status register

Sleep Mode:

flush	Flush all outgoing messages and network variable updates
sleep	Enter low-power mode by disabling system clock
timers_off	Turn off all software timers

Utilities:

abs	Arithmetic absolute value
bcd2bin	Convert Binary Coded Decimal data to binary
bin2bcd	Convert binary data to Binary Coded Decimal
max	Arithmetic maximum of two values
min	Arithmetic minimum of two values
muldiv	Multiply/divide with 32-bit intermediate result - (un)signed
random	Generate eight-bit random number
refresh_memory	Rewrite contents of EEPROM memory
reverse	Reverse the order of bits in an eight-bit number

Input/Output (see section 8 for details):

io_in	Input data from I/O object
io_out	Output data to I/O object
io_out_request	Request ready indication from parallel I/O object
io_change_init	Initialize reference value for *io_changes* event
io_select	Set timer/counter multiplexer
io_set_direction	Change direction of I/O pins
io_set_clock	Set timer/counter clock rate

7.6 Built-in Variables

input_value	Data read by last explicit or implicit *io_in()* call
input_is_new	True if last input from a timer/counter object read an updated value
nv_array_index	Index of element in network variable array with updated value
nv_in_addr	Source address of the last network variable update

8 Application I/O

The NEURON CHIP connects to application-specific external hardware via eleven pins named IO0 through IO10. These pins may be configured in numerous ways to provide flexible input and output functions with a minimum of external circuitry. Pins IO4 through IO7 have programmable (on or off) pull-ups. Pins IO0 through IO3 have high

current (20mA) sink capability. All pins (IO0 through IO10) have TTL-level inputs with hysteresis. Pins IO0 through IO7 also have low-level detect latches.

The NEURON CHIP has two 16-bit timer/counters on-chip. The input to timer/counter 1 is selectable among pins IO4 through IO7 via a programmable multiplexer (mux) and its output may be connected to pin IO0. The input to timer/counter 2 may be connected to pin IO4 and its output to pin IO1. The timer/counters are implemented as a 16-bit load register writeable by the CPU, a 16-bit counter, and a 16-bit latch readable by the CPU. The load register and latch are accessed a byte at a time. Note that no I/O pins are dedicated to timer/counter functions. If for example, timer/counter 1 is used for input signals only, then IO0 is available for other input or output functions. Timer/counter clock and enable inputs may be from external pins, or from scaled clocks derived from the system clock; the clock rates of the two timer/counters are independent of each other. External clock actions occur optionally on the rising edge, the falling edge, or both rising and falling edges of the input.

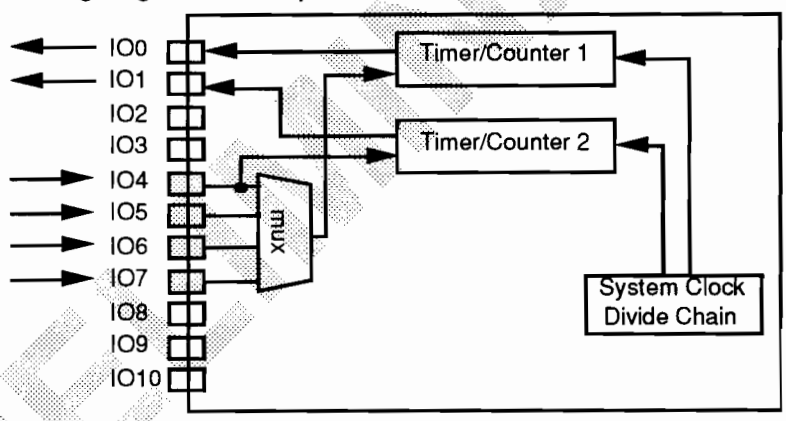


Figure 8.1 NEURON CHIP Timer/counter external connections

The programming model allows the programmer to declare one or more pins as I/O objects. The user's program may then refer to these objects in *io_in* and *io_out* system calls to perform the actual input/output operations during execution of the program. There are 24 different I/O modes available - 11 for input, 11 for output and two bidirectional modes. Certain events are associated with changes in input values. The task scheduler can thus execute associated application code when these changes occur.

I/O Mode	In/Out	Applicable I/O Pins	Input /Output Value
Bit	I/O	IO0 - IO10	0, 1 binary data
Byte	I/O	IO0 .. IO7	0 .. 255 binary data
Level Detect	I	IO0 - IO7	Logic zero level detected
Nibble	I/O	Any adjacent 4 in IO0 - IO7	0 .. 15 binary data

Table 8.1 Summary of basic I/O modes

I/O Mode	In/Out	Applicable I/O Pins	Input/Output Value
Parallel	Both	IO0 .. IO10	Parallel bidirectional handshaking port

Table 8.2 Summary of parallel I/O mode

I/O Mode	In/Out	Applicable I/O Pins	Input/Output Value
Bit Shift	I/O	Any adjacent pair	Up to 16 bits of clocked data
Neurowire	Both	IO8..IO10 + IO0 - IO7	Up to 256 bits of bidirectional serial data
Serial	I/O	IO8+IO10	8-bit characters at 600, 1200, 2400 or 4800 baud

Table 8.3 Summary of serial I/O modes

I/O Mode	Applicable I/O Pins	Input Signal
On-Time	IO4 - IO7	Pulse width of 0.2 µs - 1.678 s
Period	IO4 - IO7	Signal period of 0. 2 µs - 1.678 s
Pulse Count	IO4 - IO7	0 .. 65,535 input edges during 0.839 s
Quadrature	IO4 +IO5, IO6+IO7	± 16,383 binary Gray code transitions
Total Count	IO4 - IO7	0 .. 65,535 input edges

Table 8.4 Summary of timer/counter input modes

I/O Mode	Applicable I/O Pins	Output Signal
Frequency	IO0, IO1	Square wave of 0.3 Hz to 2.5 MHz
One-Shot	IO0, IO1	Pulse of duration 0.2 µs to 1.678 s
Pulse Count	IO0, IO1	0 .. 65,535 pulses
Pulse Width	IO0, IO1	0 .. 100 % duty cycle pulse train
Triac	IO0, IO1 + IO4 - IO7	Delay of output pulse w.r.t. input edge
Triggered Count	IO0, IO1 + IO4 - IO7	Output pulse controlled by counting input edges

Table 8.5 Summary of timer/counter output modes

8.1 Basic I/O Modes

Various combinations of I/O pins may be configured as basic inputs or outputs. The application program may optionally specify the initial values of basic outputs. Pins configured as outputs may also be read as inputs, returning the value last written.

Bit I/O

Pins IO0 through IO10 may be individually configured as single bit input or output ports. Inputs may be used to sense TTL-level compatible logic signals from external logic, contact closures, and the like. Outputs may be used to drive external CMOS-level compatible logic, and switch transistors and relays to actuate higher-current external objects such as stepper motors and lights. The high current sink capability of pins IO0 through IO3 allows these pins to drive many I/O devices directly. The direction of bit ports may be changed between input and output dynamically under application control.

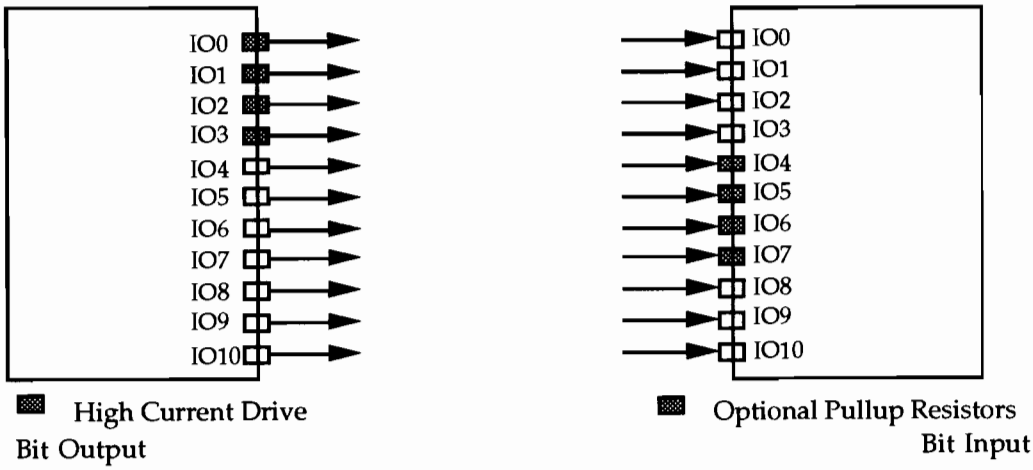


Figure 8.2 Bit I/O

Byte I/O

Pins IO0 through IO7 may be configured as a byte-wide input or output port, which may be read or written using integers in the range 0 to 255. This is useful for driving devices that require ASCII data, or other data eight bits at a time. For example, an alphanumeric display panel can use byte mode for data, and use pins IO8 - IO10 in bit mode for control and addressing. The direction of the byte port may be changed between input and output dynamically under application control.

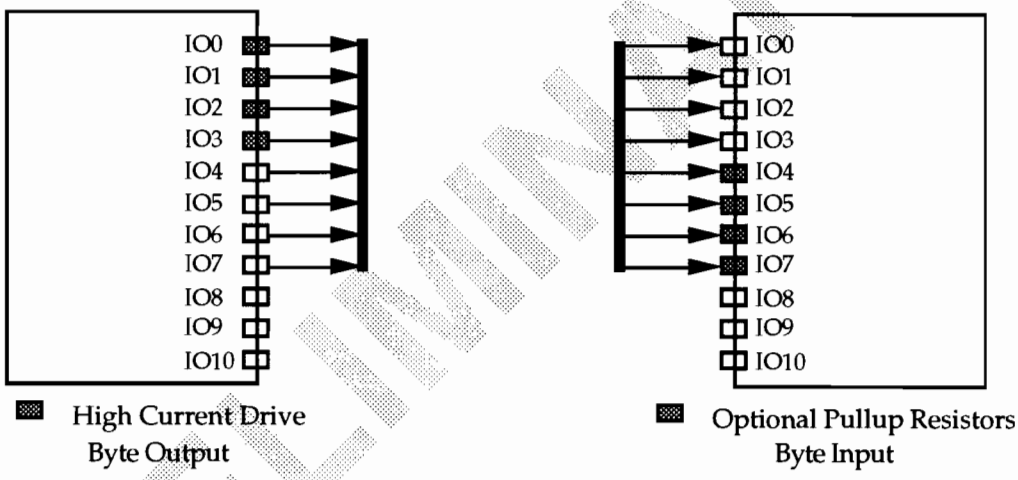


Figure 8.3 Byte I/O

Level Detect Input

Pins IO0 through IO7 may be configured as level-detect input pins, which latch a logic low level on the input pin. The application can therefore detect short pulses on the input which might be missed by software polling. This is useful for reading devices such as proximity sensors.

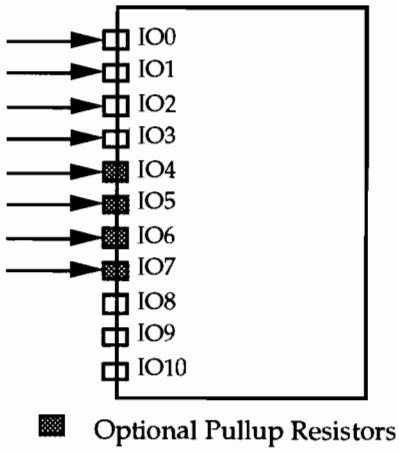


Figure 8.4 Level detect input

Nibble I/O

Groups of four consecutive pins between IO0 and IO7 may be configured as nibble-wide input or output ports, which may be read or written using integers in the range 0 to 15. This is useful for driving devices that require BCD data, or other data four bits at a time. For example, a 4 x 4 key switch matrix may be scanned by using one nibble to generate a row select output (one of four rows), and one nibble to read the input from the columns of the switch matrix. The direction of nibble ports may be changed between input and output dynamically under application control.

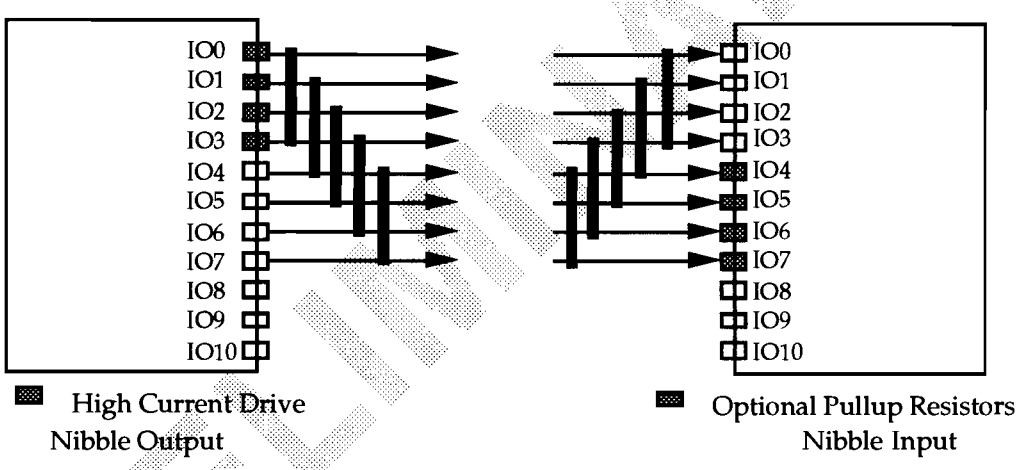


Figure 8.5 Nibble I/O

8.2 Parallel I/O

Pins IO0 through IO10 may be configured as a bidirectional 8-bit data and 3-bit control port for connecting to an external processor. The other processor may be a computer, micro-controller, or another NEURON CHIP for router, gateway or other applications.

The parallel interface on the NEURON CHIP may be configured in either master, slave A or slave B mode. In all cases, pins IO0 through IO7 are used as an eight-bit wide bidirectional data bus. When configured in master mode, the NEURON CHIP drives IO8 low as a chip select and IO9 to specify whether this is a read (high) or a write (low) cycle, and accepts IO10 as the handshake acknowledgment signal.

When configured in slave A mode, the NEURON CHIP accepts IO8 as a chip select and IO9 to specify whether this is a read or a write cycle (with respect to the master), and drives IO10 as the handshake acknowledgement signal. This is designed for use with a master processor that has an eight-bit data port plus a three-bit control port. A NEURON CHIP in master mode may be directly connected to a NEURON CHIP in slave A mode to create an application-level gateway.

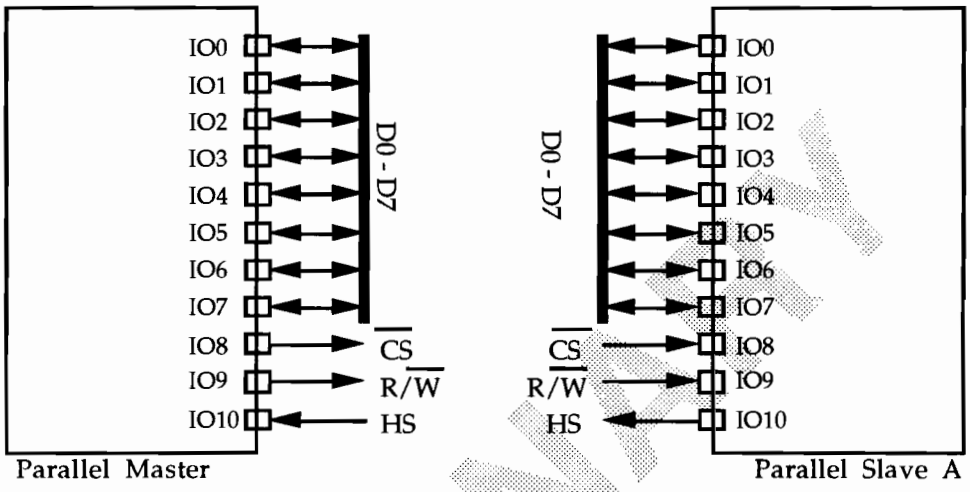


Figure 8.6 Parallel I/O - Master and Slave A

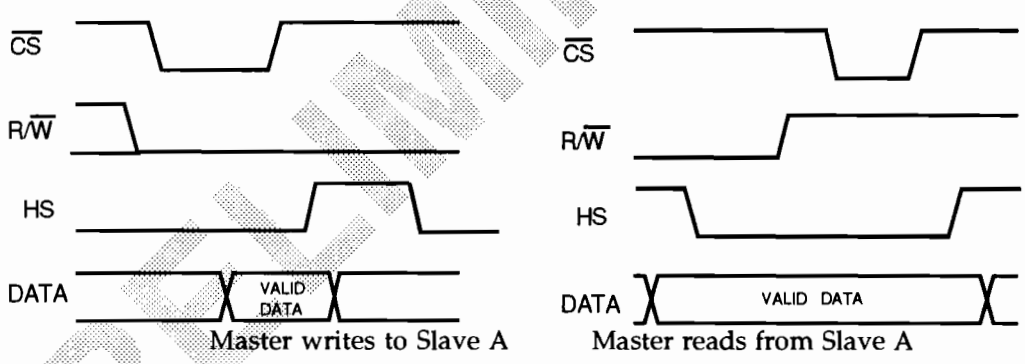
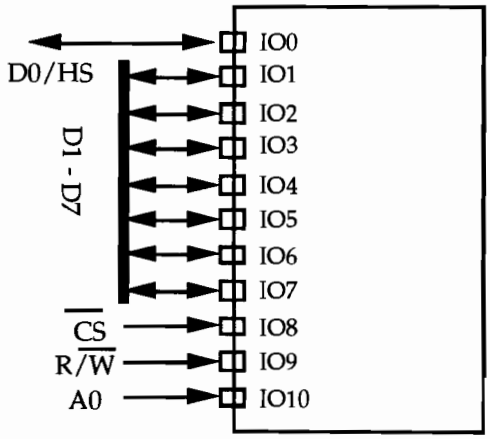


Figure 8.7 Parallel I/O: Master and Slave A signals

When configured in slave B mode, the NEURON CHIP accepts IO8 as a chip select and IO9 to specify whether the master will read or write, and accepts IO10 as a selector input. When IO10 is low and ~CS is asserted, pins IO0 through IO7 form the bidirectional data bus. When IO10 is high, R/~W is high, and ~CS is asserted, IO0 is driven as the handshake acknowledgment signal to the master. This is designed for use with a master processor that uses memory-mapped I/O. The least significant bit of the master's address bus is connected to the NEURON CHIP's IO10 pin. The NEURON CHIP appears as two registers in the master's address space, one of the registers being the read/write data register, and the other being the read-only control register. The least significant bit of the control register is the handshake (HS) bit.



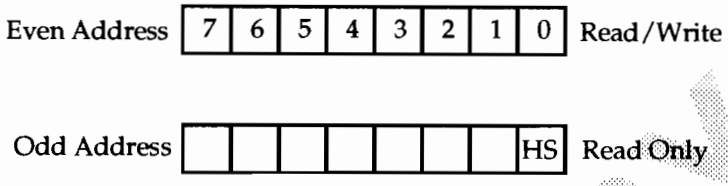


Figure 8.8 Slave B - NEURON CHIP as memory-mapped I/O device on host

The maximum data transfer rate is one byte per four CPU instruction cycles, or 2.4 µs per byte at maximum input clock rate (10 MHz). The data rate scales proportionally to the input clock rate. Hardware handshaking is used to control the instruction execution, and application processing is suspended for the duration of the transfer (up to 255 bytes per transfer). If the NEURON CHIP is waiting for parallel I/O, a watchdog timeout will occur if the transaction does not complete within 0.84 seconds (see section 9.3).

8.3 Serial I/O Modes

Bitshift I/O

Pairs of adjacent pins may be configured as serial input or output lines, the lower numbered pin being used for the clock (driven by the NEURON CHIP) and the higher numbered pin being used for up to 16 bits of serial data. The data rate may be configured as 1, 10, or 15 kbps at maximum input clock rate (10 MHz). The data rate scales proportionally to the input clock rate. The active clock edge may be specified as either rising or falling. This mode is useful for transferring data to external logic employing shift registers. This mode suspends application processing until the operation is complete.

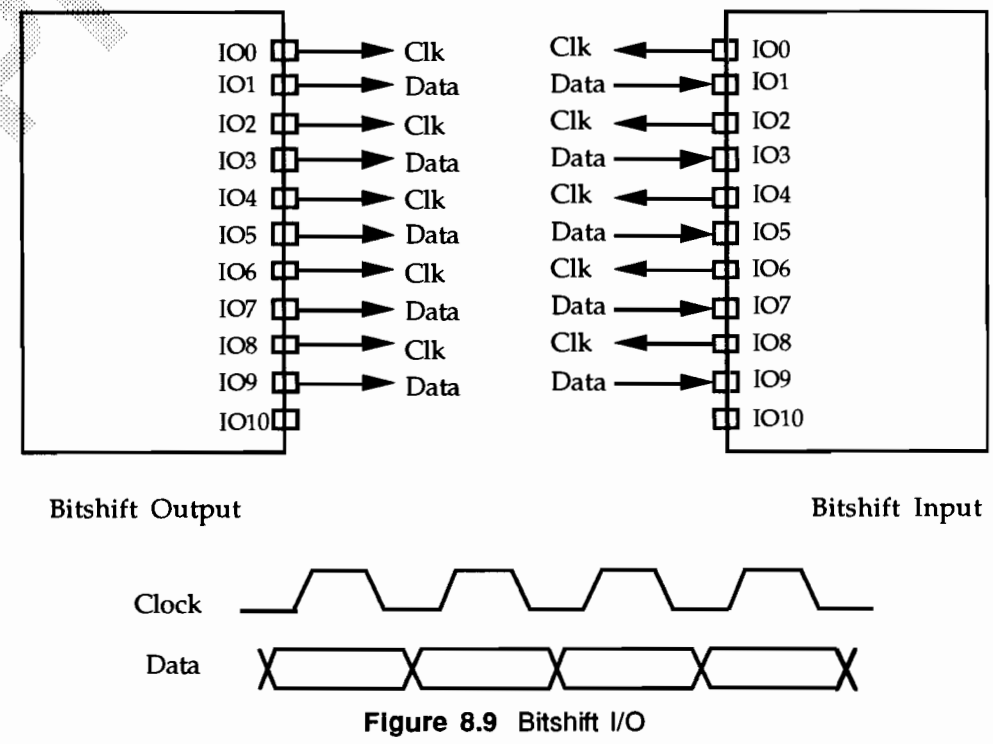


Figure 8.9 Bitshift I/O

Neurowire I/O

Pins IO8 through IO10 may be configured as a serial bidirectional port. In Neurowire master mode, pin IO8 is the clock (driven by the NEURON CHIP), IO9 is serial data output and IO10 is serial data input. Serial data is clocked out on pin IO9 at the same time as data is clocked in from pin IO10. Data is clocked by the rising edge of the clock signal. In addition, one or more of the pins IO0 through IO7 may be used as a chip select, allowing multiple Neurowire devices to be connected on a 3-wire bus. The clock rate may be specified as 1, 10 or 20 kbit/sec at an input clock rate of 10 MHz; these scale proportionally with input clock.

In Neurowire slave mode, pin IO8 is the clock (driven by the external master), IO9 is serial data output and IO10 is serial data input. Serial data is clocked out on pin IO9 at the same time as data is clocked in from pin IO10. Data is clocked by the rising edge of the clock signal, which may be up to 18 kbit/sec. One of the pins IO0 through IO7 may be designated as a time-out pin. A logic one level on the time-out pin causes the Neurowire slave I/O operation to be terminated before the specified number of bits has been transferred. This prevents the NEURON CHIP watchdog timer from resetting the chip in the event that fewer than the requested number of bits are transferred by the external clock.

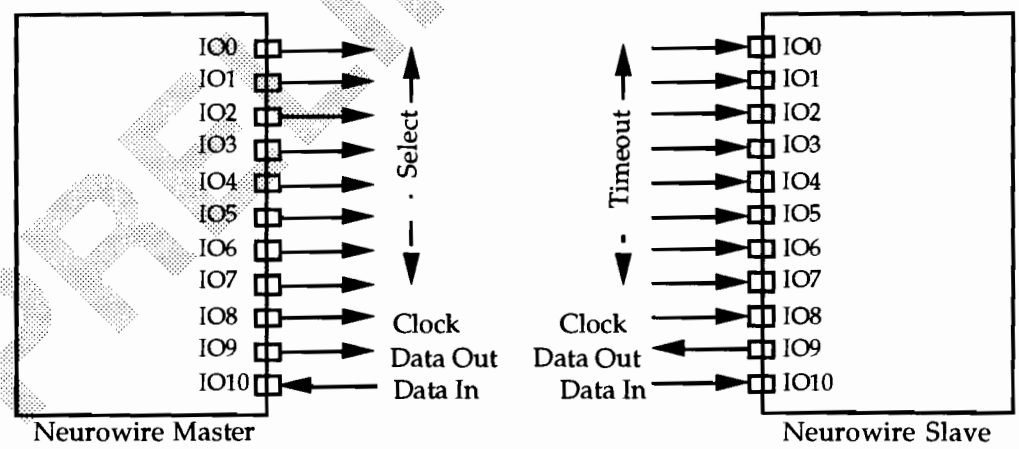


Figure 8.10 Neurowire I/O

In both master and slave modes, up to 255 bits of data may be transferred at a time. Neurowire I/O suspends application processing until the operation is complete. Neurowire mode is useful for external devices, such as A/D, D/A converters and display drivers incorporating serial interfaces that conform with National Semiconductor's Microwire™ or Motorola's SPI interface.

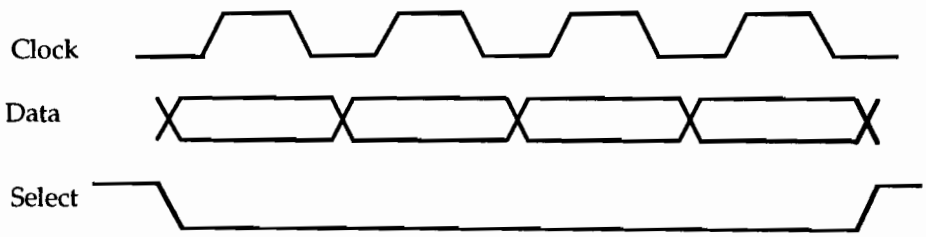


Figure 8.11 Neurowire I/O

Serial I/O

Pin IO8 may be configured as an asynchronous serial input line, and pin IO10 may be configured as an asynchronous serial output line. The bit rates for input and for output may be independently specified to be 600, 1200, 2400 or 4800 bps at maximum input clock rate (10 MHz). The data rate scales proportionally to the input clock rate. The frame format is fixed at one start bit, eight data bits and one stop bit, and up to 255 bytes may be transferred at a time. Either a serial input or a serial output operation may be in effect at any one time, but not both - the interface is half-duplex only. This mode suspends application processing until the operation is complete. On input, the *io_in* request will time out after 20 character times if no start bit is received. If the stop bit has the wrong polarity (it should be a one), the input operation is terminated with an error. The application code can use bit I/O pins for flow control handshaking if required. This mode is useful for transferring data to and from serial devices such as terminals, modems, and computer serial interfaces.

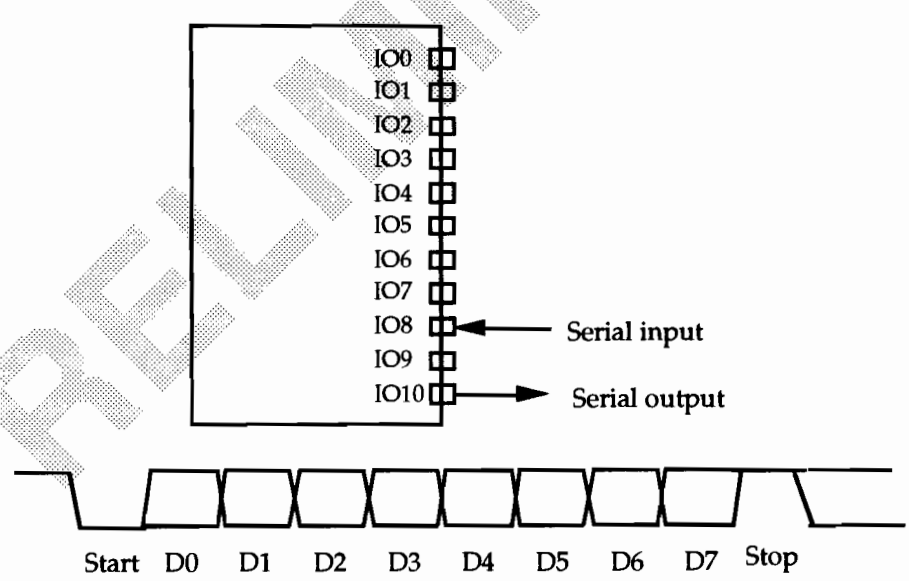


Figure 8.12 Serial I/O

8.4 Timer/Counter Interface

The NEURON CHIP has two 16-bit timer/counters. For the first timer/counter, IO0 is used as the output, and a multiplexer selects one of IO4 through IO7 as the input. The second timer/counter uses IO4 as the input, and IO1 as the output (see figure 8.1). Note that multiple timer/counter input objects may be declared on different pins within a single application. By calling *io_select,* the application can use the first timer/counter in up to four different input modes. If a timer/counter is configured in one of the output modes, or configured as a quadrature input, then it cannot be re-assigned to another timer/counter function in the same application program.

Timer/Counter Input Modes

On-Time Input

A timer/counter may be configured to measure the time for which its input is asserted. Table 8.6 shows the resolution and maximum times for different I/O clock selections. Assertion may be defined as either logic high or logic low. This mode may be used in conjunction with a simple analog-to-digital converter with a voltage-to-time circuit, or for measuring velocity by timing motion past a position sensor.

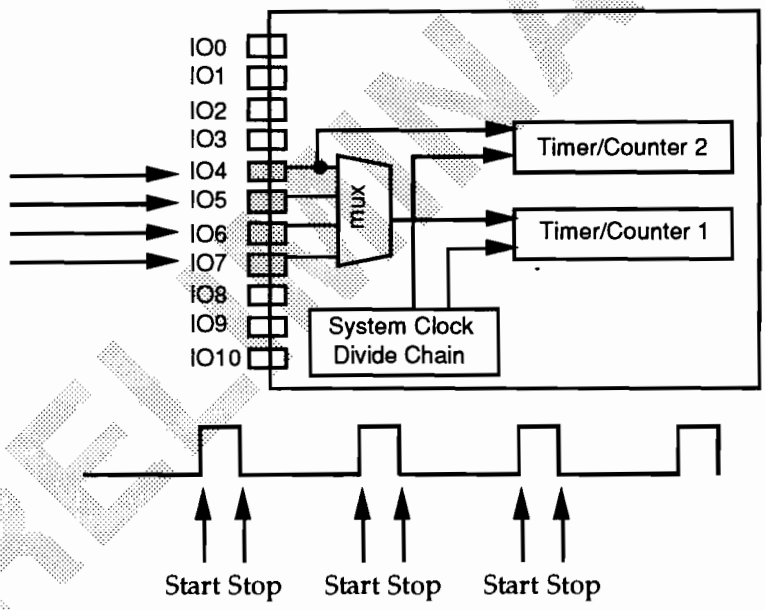
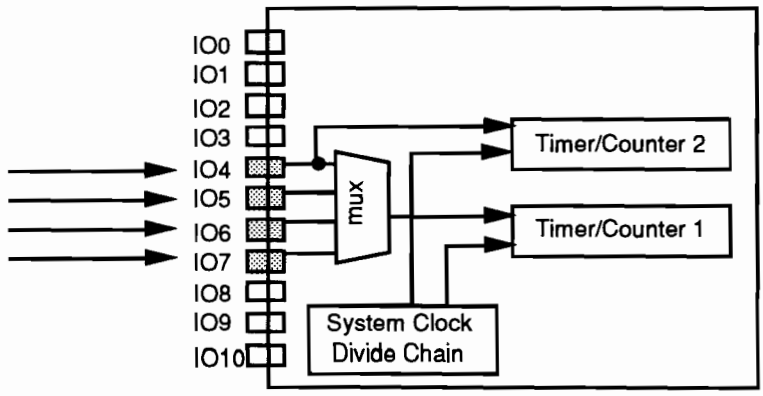


Figure 8.13 On-time input

Period Input

A timer/counter may be configured to measure the period from one rising or falling edge to the next corresponding edge on the input. Table 8.6 shows the resolution and maximum time measured for various clock selections. This mode is useful for instantaneous frequency or tachometer applications. Analog-to-digital conversion can be implemented using a voltage-to-frequency converter with this mode.



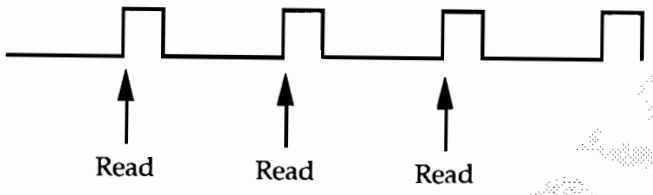


Figure 8.14 Period input

Pulse Count Input

A timer/counter may be configured to count the number of input edges (up to 65,535) in a fixed time (0.8388608 second) at all allowed input clock rates. Edges may be defined as rising or falling. The measurement period is not synchronized with the input waveform. This mode is useful for average frequency measurements, or tachometer applications.

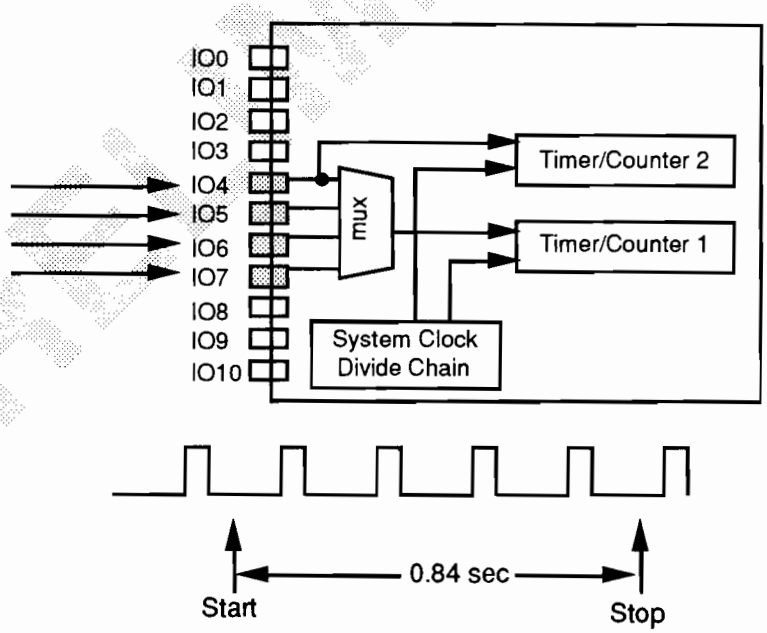


Figure 8.15 Pulse count input

Quadrature Input

A timer/counter may be configured to count transitions of a binary Gray code input on two adjacent input pins. The Gray code is generated by devices such as shaft encoders and optical position sensors which generate the bit pattern (00,01,11,10,00...) for one direction of motion and (00,10,11,01,00...) for the opposite direction. Reading the value of a quadrature object gives the arithmetic net sum of the number of transitions since the last time it was read (-16,384 to 16,383). The maximum frequency of the input is one quarter of the input clock rate, for example 2.5 MHz at the maximum 10 MHz NEURON CHIP input clock. Quadrature devices may be connected to timer/counter 1 via pins IO6 and IO7, and timer/counter 2 via pins IO4 and IO5.

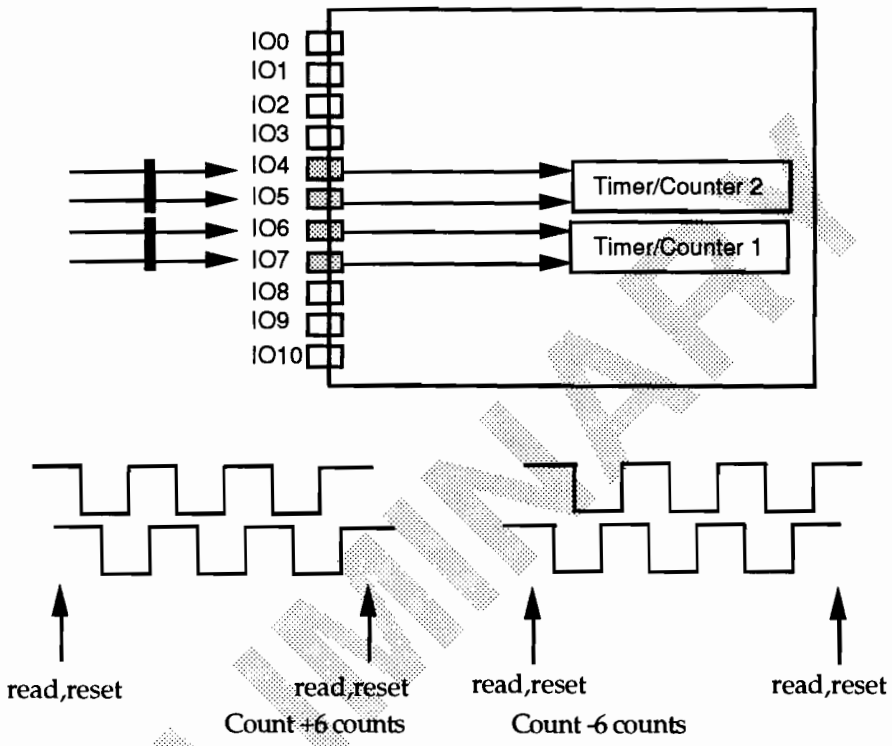


Figure 8.16 Quadrature input

Total Count Input

A timer/counter may be configured to count input edges, either rising or falling but not both. Reading the value of a total count object gives the number of transitions since the last time it was read (0 to 65,535). Maximum frequency of the input is one quarter of the input clock rate, for example 2.5 MHz at the maximum 10 MHz NEURON CHIP input clock. This mode is useful for counting external events such as contact closures, where it is important to keep an accurate running total.

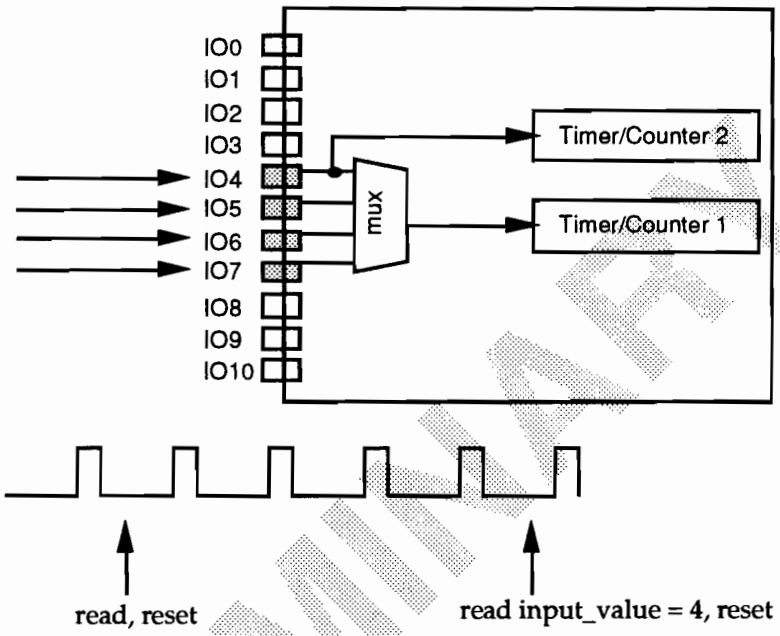


Figure 8.17 Total count input

8.4.2 Timer/Counter Output Modes

Frequency Output

A timer/counter may be configured to generate a continuous square wave of 50% duty cycle. The resolution and maximum value of the half-period of the signal is given by table 8.8. Writing a new frequency value to the device takes effect at the end of the current cycle. This mode is useful for frequency synthesis to drive an audio transducer, or to drive a frequency to voltage converter to generate an analog output.

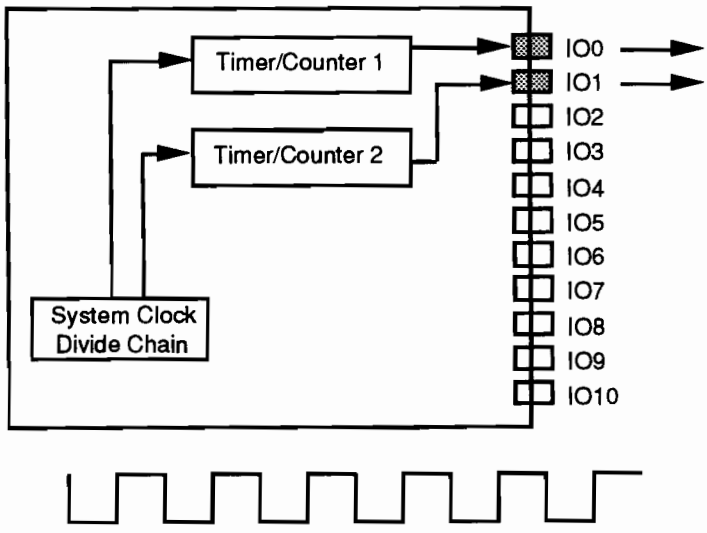


Figure 8.18 Frequency output

One-Shot Output

A timer/counter may be configured to generate a single pulse of programmable duration. The asserted state may be either logic high or logic low. Retriggering the one-shot before the end of the pulse causes it to continue for the new duration. Table 8.6 gives the resolution and maximum time of the pulse for various clock selections. This mode is useful for generating a time delay without intervention of the application CPU.

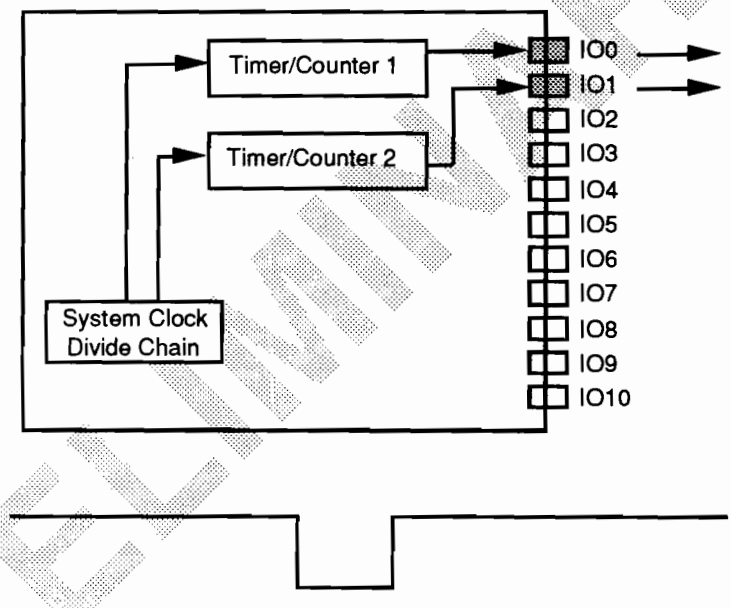


Figure 8.19 One-shot output

Pulse Count Output

A timer/counter may be configured to generate a series of pulses. The number of pulses output is in the range 0 to 65,535, and the output waveform is a square wave of 50% duty cycle. This mode suspends application processing until the pulse train is complete. The frequency of the waveform may be one of eight values given by table 8.7, with clock select values of 1 through 7. This mode is useful for external counting devices that can accumulate pulse trains, such as stepper motors.

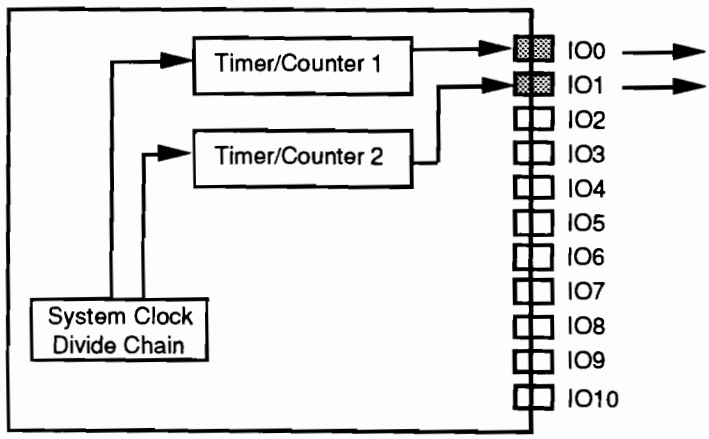


Figure 8.20 Pulse count output

Pulse Width Output

A timer/counter may be configured to generate a pulse-width modulated repeating waveform. In pulsewidth short mode, the duty cycle ranges from 0% to 100% (0/256 to 256/256) of a cycle in steps of about 0.4% (1/256). In pulsewidth short mode, the frequency of the waveform may be one of the eight values given by table 8.7. In pulsewidth long mode, the duty cycle ranges from 0% to almost 100%(0/65,536 to 65,535/65,536) of a cycle in steps of 15.25 ppm (1/65,536). In pulsewidth long mode, the frequency of the waveform may be one of the eight values given by table 8.8. The asserted state of the waveform may be either logic high or logic low. Writing a new pulsewidth value to the device takes effect at the end of the current cycle. A pulsewidth modulated signal provides a simple means of digital to analog conversion.

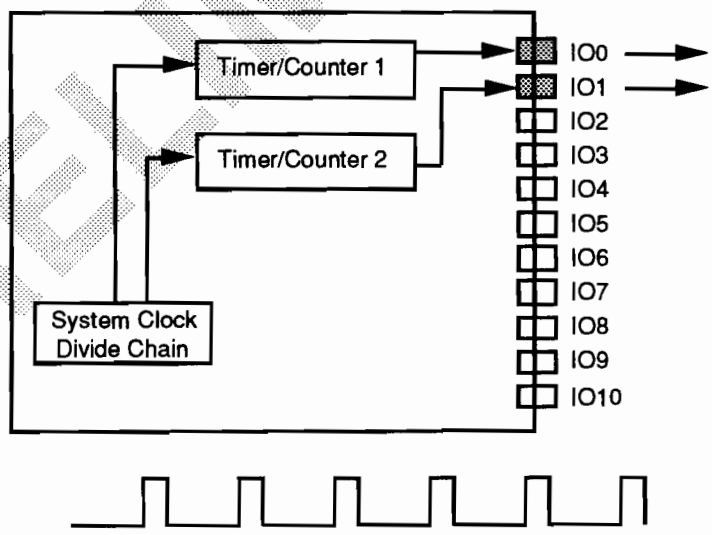


Figure 8.21 Pulse width output

Triac Output

On the NEURON 3150 CHIP, a timer/counter may be configured to control the delay of an output pulse signal with respect to either a rising edge or a falling edge as a synchronization input. On the NEURON 3120 CHIP, this sync input may be either a rising edge, a falling edge, or both rising and falling edges. The output pulse is 25 μs wide, independent of the NEURON CHIP input clock. For control of AC circuits using a triac device, the sync input is typically a zero-crossing signal, and the pulse output is the triac trigger signal. Table 8.6 shows the resolution and maximum range of the delay.

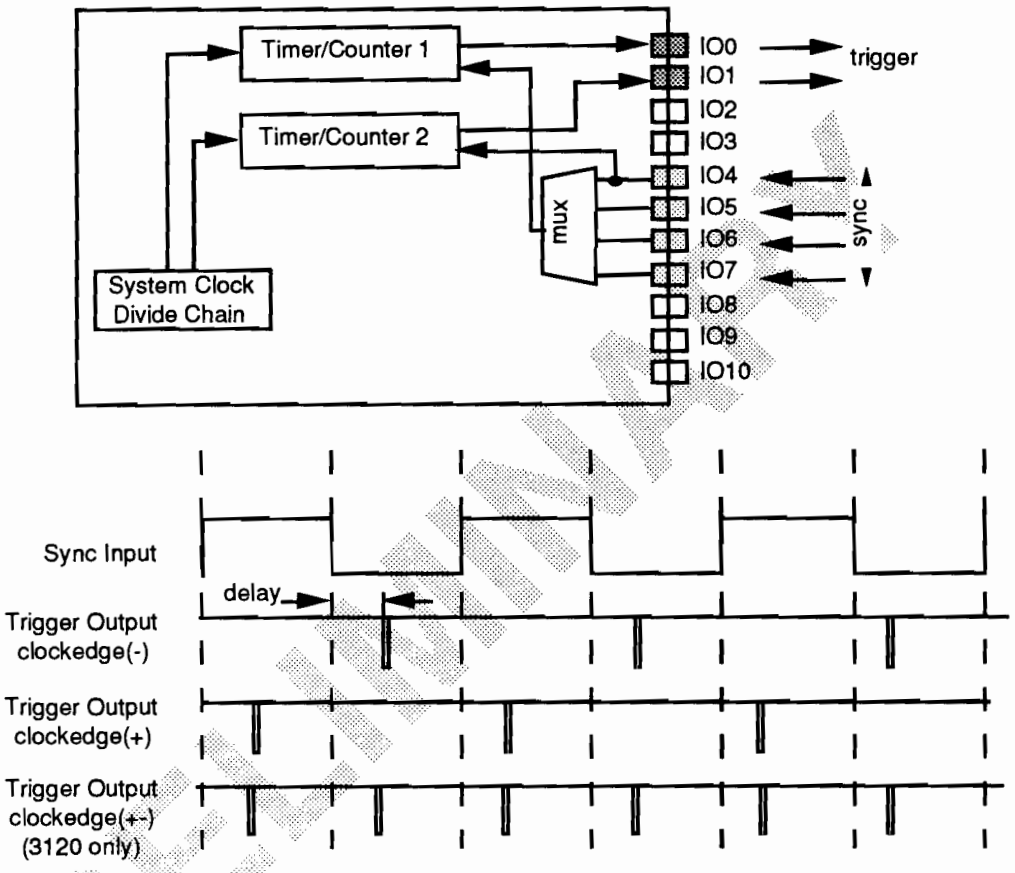


Figure 8.22 Triac output

Triggered Count Output

A timer/counter may be configured to generate an output pulse that is asserted under program control, and de-asserted when a programmable number of input edges (up to 65,535) has been counted on a second pin. Assertion may be either logic high or logic low. This mode is useful for controlling stepper motors or positioning actuators which provide position feedback in the form of a pulse train. The drive to the device is enabled until it has moved the required distance, and then the device is disabled.

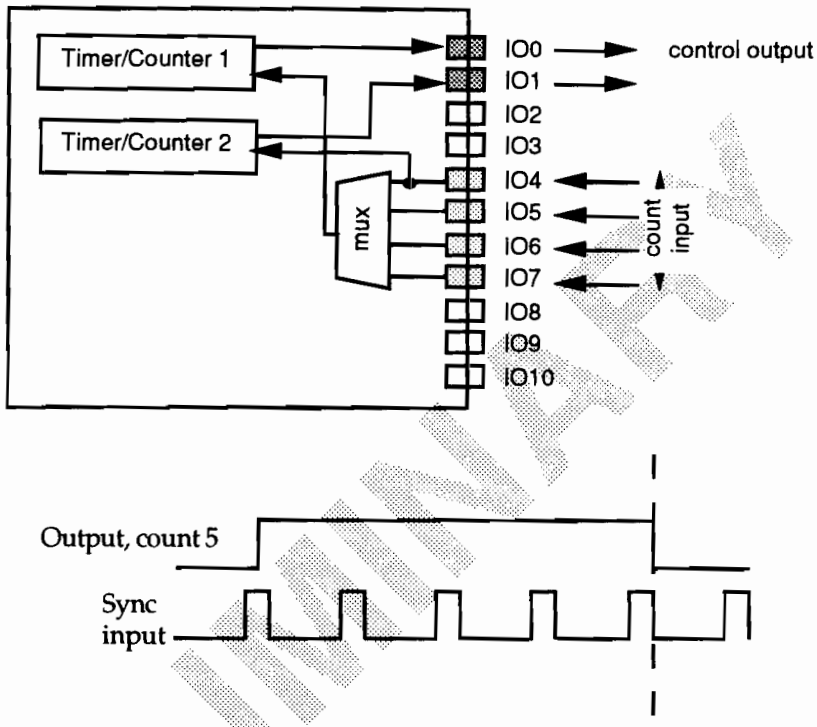


Figure 8.23 Triggered count output

Notes

For on-time input and period input, the timer/counter returns a number in the range 0 to 65,535, representing elapsed times from zero up to the maximum range given in table 8.6.

For one-shot output, frequency output, and triac output, the timer/counter may be programmed with a number in the range 0 to 65,535, representing waveform on-times from zero up to the maximum range given in table 8.6. The clock select value is specified in the declaration of the I/O object in the NEURON C application program, and may be modified at runtime.

Clock Select	Resolution (µs)	Maximum Range (ms)
0	0.2	13.11
1	0.4	26.21
2	0.8	52.42
3	1.6	104.86
4	3.2	209.71
5	6.4	419.42
6	12.8	838.85
7	25.6	1,677.70

Table 8.6 Timer/counter resolution and maximum range

This table is for 10 MHz input clock. Scale appropriately for other clock rates:

$$\text{Resolution } (\mu s) = 2^{(\text{Clock Select} + 1)} / \text{ Input Clock (MHz)}$$

$$\text{Maximum Range (ms)} = 65.535 \times \text{Resolution } (\mu s)$$

For pulse width short output and pulse count output, table 8.7 gives the possible choices for pulse train repetition frequencies, except that pulse count output may not be used with clock select 0.

Clock Select	Frequency (Hz)	Period (µs)
0	19,531	51.2
1	9,766	102.4
2	4,883	204.8
3	2,441	409.6
4	1,221	819.2
5	610	1,638.4
6	305	3,276.8
7	153	6,553.6

Table 8.7 Timer/counter pulse train output

This table is for 10 MHz input clock. Scale appropriately for other clock rates:

Period (µs) = 512 x $2^{\text{Clock Select}}$ / Input Clock (MHz)

Frequency (Hz) = 1,000,000 / Period (µs)

For pulse width long output, table 8.8 gives the possible choices for pulse train repetition frequencies.

Clock Select	Frequency (Hz)	Period (ms)
0	76.29	13.1072
1	38.15	26.2144
2	19.07	52.4288
3	9.54	104.8576
4	4.77	209.7152
5	2.38	419.4304
6	1.19	838.8608
7	0.60	1,677.7216

Table 8.8 Timer/counter pulse train output

This table is for 10 MHz input clock. Scale appropriately for other clock rates:

Period (ms) = 131.072 x $2^{\text{Clock Select}}$ / Input Clock (MHz)

Frequency (Hz) = 1,000 / Period (ms)

9 Additional Functions

9.1 Service Pin

The service pin is used during configuration, installation and maintenance of a LONWORKS node. The pin has both output and input functions. As an output, the service pin is driven active-low to light an external LED. The LED is lit when the node has no valid application code or on-chip failure. The LED blinks at a 1/2 Hz rate when the node has not been configured with network address information. A logic-low input at the service pin causes the NEURON CHIP to transmit a network management message containing its 48-bit NEURON ID on the network. To accomplish both functions, the pin

is multiplexed between input and output at a 76 Hz rate with a 50% duty cycle (see figure 9.1). The service pin has an optional on-chip pull-up to bring the input to an inactive-high state for use when the LED and pull-up resistor are not connected.

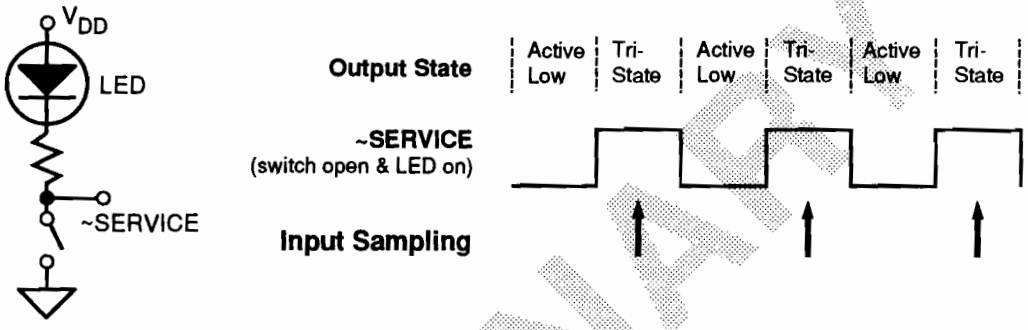


Figure 9.1 Service pin circuit

9.2 Sleep/Wake-up Circuitry

The NEURON CHIP may be put into a low-power sleep mode under software control. In this mode, the oscillator, system clock and all timer/counters are turned off, but all state information (including the contents of on-chip RAM) is retained. Normal operation resumes when an input transition occurs on any one of the following pins:

- I/O pins (maskable):
  One of IO4 through IO7, selected by the timer/counter multiplexer
- Service pin (not maskable)
- Communications port (maskable):
  Single-Ended Mode - CP0
  Differential Mode - CP0 or CP1
  Special Purpose mode - CP3

The application software may optionally specify that the programmable pull-ups on the service pin and I/O pins IO4 - IO7 be disabled to further reduce power consumption.

9.3 Watchdog Timer

The NEURON CHIP CPUs are protected against malfunctioning software or memory faults by three watchdog timers (one per CPU). If application or system software fails to reset these timers periodically, the entire NEURON CHIP is automatically reset. The watchdog period is approximately 0.84 seconds at maximum input clock rate (10 MHz) and scales inversely with the input clock rate. When the NEURON CHIP is in sleep mode, all the watchdog timers are disabled.

9.4 Reset Circuitry

The RESET pin is an open-drain active low input to the NEURON CHIP, with an internal pull-up. When power is applied to the following circuit, the reset capacitor C_r charges via the internal pull-up and the diode, providing a clean reset to the NEURON CHIP and other attached circuitry. When V_{DD} is turned off, the circuit resets the NEURON CHIP by discharging the capacitor through the diode and the source impedance of the power supply.

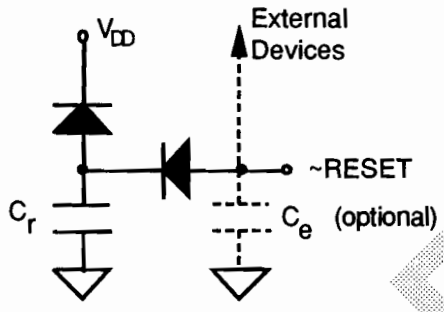


Figure 9.2 External reset circuit

The value of the capacitance is given by:

$C_r = (300\ \mu A + I) \times T / V$

where:

C_r = minimum value of reset capacitance
300 μA = maximum built-in pull-up current
I = maximum pull-up current due to external devices (e.g. TTL input current)
T = time for V_{DD} to reach 90% of final value
V = final V_{DD} voltage

For example, if a 5.25 V power supply ramps up in one full 60 Hz cycle (16.67 msec), and external components could contribute up to 15 μA of pull-up current, a minimum value of 1 μF capacitance is required.

The RESET pin may also be activated as an output under software control. It may therefore be used to reset external circuitry such as transceivers. In this case, the optional external reset capacitor C_e is needed.

$C_e = (300\ \mu A + I) \times T / (V - 0.5\ v)$

where:

C_e = minimum value of external reset capacitance
300 μA = maximum built-in pull-up current
I = maximum pull-up current due to external circuits
T = reset pin low hold time for external devices to reset properly
V = minimum reset V_{il} level of external device
0.5 V = maximum expected offset of RESET pin

For example, if two external LSTTL devices (with V_{il} = 0.8 V) contribute 400 μA each of pull-up current, and require a 30 nsec reset low hold time, a total minimum value of 110 pF capacitance is required. The total capacitance directly connected to the RESET pin, including stray and external device input capacitance, may not exceed 250 pF. A NEURON CHIP needs no capacitance on the RESET pin to reset itself or be reset by an external device.

It is important to hold the ~RESET pin low whenever V_{DD} is below its recommended operating condition to avoid any possible EEPROM corruption during power-down (see section 4.1).

9.5 Clocking System

The NEURON CHIP includes an oscillator that may be used to generate an input clock using an external crystal or ceramic resonator circuit. The transconductance of this

oscillator is 2.1 milli-Siemens minimum. The NEURON CHIP may operate over a range of input clock rates from 10 MHz down to 625 kHz for low power applications. The valid input clock frequencies are 10 MHz, 5 MHz, 2.5 MHz, 1.25 MHz, and 625 kHz. Alternatively, an externally generated clock may drive the CMOS input pin CLK1 of the NEURON CHIP, in which case CLK2 should be left unconnected. The accuracy of the input clock frequency must be ±1.5% or better (with temperature and voltage variations) to ensure that nodes correctly communicate on the network.

The NEURON CHIP divides the input clock by a factor of two to provide a symmetrical on-chip system clock. The system clock is further divided by powers of two to provide clocks for the applications I/O section, the network communications port and the CPU watchdog timers.

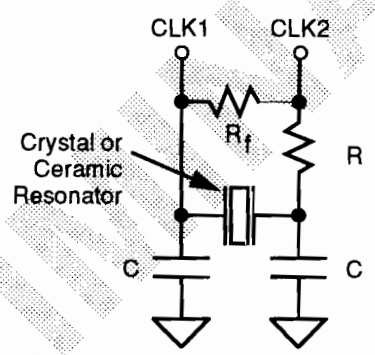


Figure 9.3 NEURON CHIP clock generator circuit

Input Clock Frequency	Crystal		Ceramic Resonator	
	R	C	R	C
10.0 MHz	270 Ω	30 pF	270 Ω	30 pF
5.0 MHz	470 Ω	30 pF	270 Ω	30 pF
2.5 MHz	1 KΩ	36 pF	1 KΩ	36 pF
1.25 MHz	1.2 KΩ	47 pF	1.2 KΩ	47 pF
625 kHz	2.7 KΩ	47 pF	1.2 KΩ	100 pF

Table 9.1 Clock generator component values

R_f = 100 KΩ for all conditions. Crystal or ceramic resonator frequency = Input clock frequency.

Crystal may be parallel or series resonant. NPO-type ceramic capacitors are recommended. Resistor and capacitor tolerance is ±5%. Table 9.2 lists a source for the ceramic resonators as an aid in the selection of components.

Supplier	Murata Erie 2200 Lake Park Drive Smyrna, Georgia 30080 Tel: 404-436-1300
10 MHz	CSA 10.0 MT
5 MHz	CSA 5.0 MT
2.5 MHz	CSA 2.5 MG

Table 9.2 Supplier of ceramic resonators

10 NEURON 3150 CHIP Pin Assignment and Pad Layout

Pin Mnemonic	Characteristics	Pin Number
IO0 - IO3	TTL input; high sink output	2 - 5
IO4 - IO7	TTL input, configurable pull-up; standard output	10 - 13
IO8 - IO10	TTL input; standard output	14 - 16
CP0 - CP1	differential or TTL input; standard output	28 - 29
CP2 - CP3	TTL input; high drive output	30 - 31
CP4	TTL input; standard output	32
CLK1	CMOS input	24
CLK2	special output (for use in oscillator configuration)	23
~RESET	TTL input; high sink open drain output; pull-up	6
~SERVICE	TTL input, configurable pull-up; high sink output	17
A0 - A14, A15	standard output	64 - 50, 47
D0 - D1, D2 - D7	TTL input; standard output	43 - 42, 38 - 33
~E	standard output	46
R/~W	standard output	45
V_{SS}	Ground	8,9,19,21,25,39
V_{DD}	Power	7,20,22,26,40,41,44
NC	Not connected	1,18,27,48,49

Table 10.1 NEURON 3150 CHIP pin assignments

Note: Bypass capacitors should be placed close to the V_{DD} pins 7, 22, 26, 40 and 41.

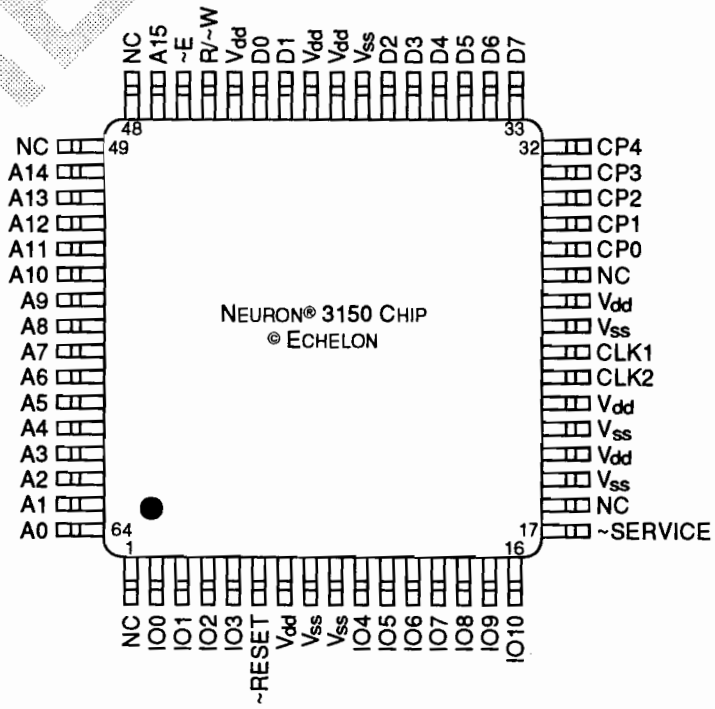


Figure 10.1 NEURON 3150 CHIP pin assignment (top view)

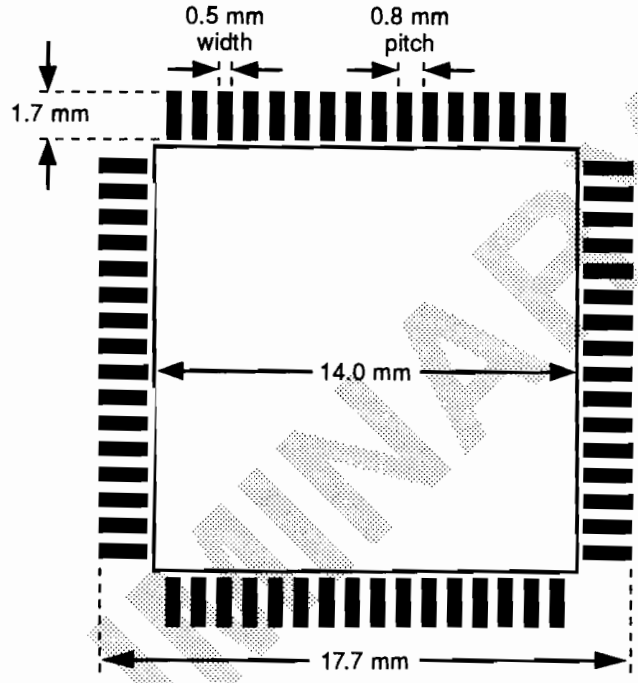


Figure 10.2 Recommended pad layout for NEURON 3150 CHIP

11 NEURON 3120 CHIP Pin Assignment and Pad Layout

The following information is preliminary. Before beginning any design work, contact Motorola or Toshiba for current specifications.

Pin Mnemonic	Characteristics	Pin Number
IO0 - IO3	TTL input; high sink output	7, 6, 5, 4
IO4 - IO7	TTL input, configurable pull-up; standard output	3, 30, 29, 28
IO8 - IO10	TTL input; standard output	27, 26, 24
CP0 - CP1	differential or TTL input; standard output	19, 20
CP2 - CP3	TTL input; high drive output	17, 21
CP4	TTL input; standard output	22
CLK1	CMOS input	15
CLK2	special output (for use in oscillator configuration)	14
~RESET	TTL input; high sink open drain output; pull-up	1
~SERVICE	TTL input, configurable pull-up; high sink output	8
V_{SS}	Ground	9, 10, 13, 16, 23, 31
V_{DD}	Power	2, 11, 12, 18, 25, 32

Table 11.1 NEURON 3120 CHIP pin assignment

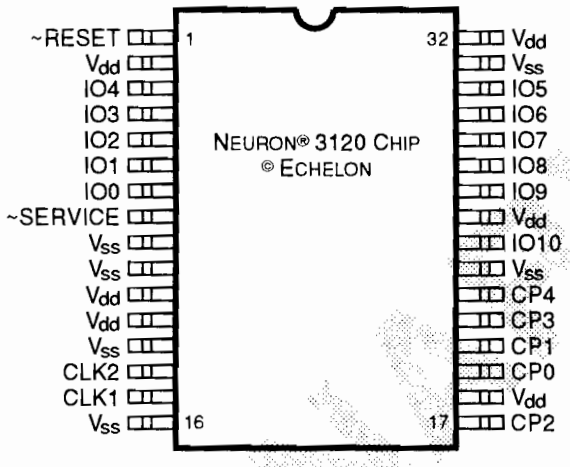


Figure. 11.1 NEURON 3120 CHIP pin assignment (top view)

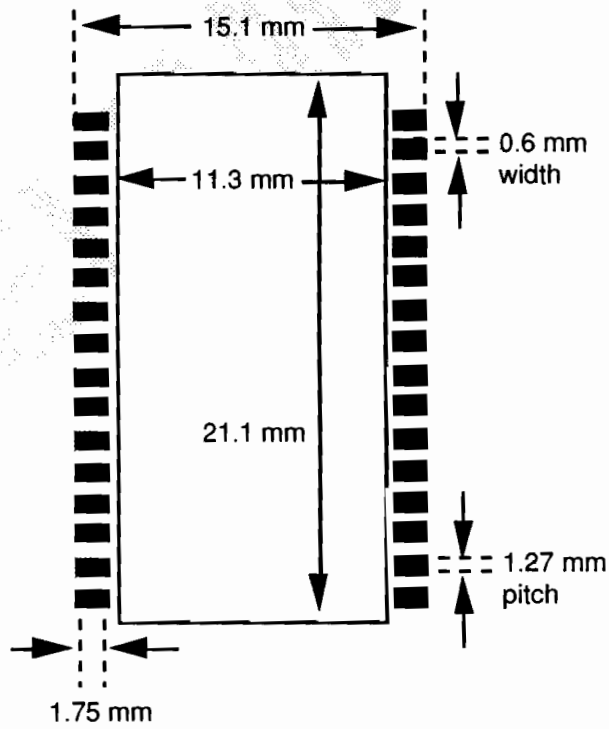


Figure 11.2 Recommended pad layout for NEURON 3120 CHIP

12 Electrical and Environmental Specifications

Specifications are over operating free-air temperature range unless otherwise noted. This information is preliminary. Refer to Motorola's and Toshiba's individual data sheets for the latest specifications.

12.1 General

Supply voltage range, V_{DD}	-0.3 to 7.0 V
Input voltage range	-0.3 to V_{DD} + 0.3 V
Maximum source current, I_{DD}	200mA
Maximum drain current, I_{SS}	300 mA
Continuous power dissipation	1 watt
Storage temperature range	-65 to 150 °C

Table 12.1 Absolute Maximum Ratings

Supply voltage	4.5 to 5.5 V with respect to V_{SS}
TTL Low-level input voltage	V_{SS} to 0.8 V
TTL High-level input voltage	2.0 V to V_{DD}
Standard Low-level input voltage	V_{SS} to 0.8 V
CMOS High-level input voltage	V_{DD} - 0.8 V to V_{DD}
Operating free-air temperature	-40°C to +85°C

Table 12.2 Recommended Operating Conditions

Low-level output voltage	
Standard (excludes CLK2)	0.4 V max at I_{ol} = 1.4 mA
High Sink (IO0 - IO3, ~SERVICE, ~RESET)	0.4V max at I_{ol} = 10 mA 0.8V max at I_{ol} = 20 mA
High Drive (CP2, CP3)	0.4V max at I_{ol} = 15 mA 1.0V max at I_{ol} = 40 mA
High-level output voltage (not applicable to open drain outputs)	
Standard(excludes CLK2, ~RESET)	(V_{DD} - 0.4V) min at I_{oh} = -1.4 mA
High Sink (IO0 - IO3, ~SERVICE)	(V_{DD} - 0.4 V) min at I_{oh} = -1.4 mA
High Drive (CP2, CP3)	(V_{DD} - 0.4 V) min at I_{oh} = -15 mA (V_{DD} - 1.0V) min at I_{oh} = -40 mA
Hysteresis (excluding CLK1, ~RESET)	175 mV min
Input current (excluding pull-ups)	±10 µA max over $V_{SS}<V_{in}<V_{DD}$
Pull-up source current (IO4-IO7, ~SERVICE if configured, ~RESET)	30 µA min, 300 µA max at V_o = 0 V with output high-Z
Supply current - operating mode	
Supply current - sleep mode	

Table 12.3 Electrical Characteristics

12.2 Communications Port Differential Receiver Electrical Characteristics

Specifications are over operating free-air temperature range unless otherwise noted. This information is preliminary. Refer to Motorola's and Toshiba's individual data sheets for the latest specifications.

	Min	Max
Common mode range (hysteresis as specified below)	1.2V	V_{DD}-2.2V
Common mode range (unspecified hysteresis)	0.9V	V_{DD}-1.75 V
Input offset voltage	-0.05*V_h-35mV	0.05*V_h+35mV
Propagation delay (F = 0, V_{id}=V_h/2 + 200mV)		230 ns
Input resistance	5 MΩ	
Wake up time		10 µs
Filter asymmetry (t_{plh}/t_{phl})	0.7	1.42

Table 12.4 Miscellaneous characteristics

Hysteresis	$V_h$min	V_h typ	V_h max
0	0.019V_{DD}	0.027V_{DD}	0.035V_{DD}
1	0.040V_{DD}	0.054V_{DD}	0.068V_{DD}
2	0.061V_{DD}	0.081V_{DD}	0.101V_{DD}
3	0.081V_{DD}	0.108V_{DD}	0.135V_{DD}
4	0.101V_{DD}	0.135V_{DD}	0.169V_{DD}
5	0.121V_{DD}	0.162V_{DD}	0.203V_{DD}
6	0.142V_{DD}	0.189V_{DD}	0.236V_{DD}
7	0.162V_{DD}	0.216V_{DD}	0.270V_{DD}

Table 12.5 Hysteresis values expressed as differential peak-to-peak voltages in terms of V_{DD}

Filter (F)	Min	Typ	Max
0	2	6	9 nsec
1	90	270	580 nsec
2	200	535	960 nsec
3	410	1070	1920 nsec

Table 12.6 Post-hysteresis filter values expressed as transient pulse suppression times

Specifications are over operating free-air temperature range unless otherwise noted. This information is preliminary. Refer to Motorola's and Toshiba's individual data sheets for the latest specifications.

Symbol	Parameter	Min	Max	Unit
t_{CYC}	Memory cycle time (system clock period)	200	3200	ns
PW_{EH}	Pulse width ~E High	t_{CYC}/2 - 5	t_{CYC}/2 +5	ns
PW_{EL}	Pulse width ~E Low	t_{CYC}/2 - 5	t_{CYC}/2 +5	ns
t_{AD}	Delay, ~E High to address valid	-	38	ns
t_{AH}	Address hold time	5	-	ns
t_{RD}	Delay, ~E high to R/~W valid Read	-	25	ns
t_{RH}	R/~W hold time Read	5	-	ns
t_{DSR}	Read data setup time	83	-	ns
t_{DZR}	Delay data bus high-Z to R/~W High	5	-	ns

t_{DHR}	Data hold time Read	0	-	ns
t_{WR}	Delay, ~E high to R/~W valid write	-	25	ns
t_{WDD}	Delay, R/~W Low to data drivers On	10	-	ns
t_{DDW}	Delay, ~E Low to data valid	-	67	ns
t_{WH}	R/-W hold time Write	5	-	ns

Notes: 1. t_{cyc} = 2*1/F, where F is the input clock frequency.

2. See figure 12.1. Since the data bus goes high-Z at the beginning of a read cycle, the effective data hold time is dependent on the current load on the bus. Typically this will be >> 50 ns.

3. See figure 12.1. The NEURON CHIP drives the previously read data until after the falling edge of ~E. Therefore, an external memory and the NEURON CHIP may both be driving the data lines to the same levels during this time without contention.

Table 12.7 NEURON 3150 CHIP memory timing over voltage (4.5 to 5.5 V) and temperature (0° to 70°C)

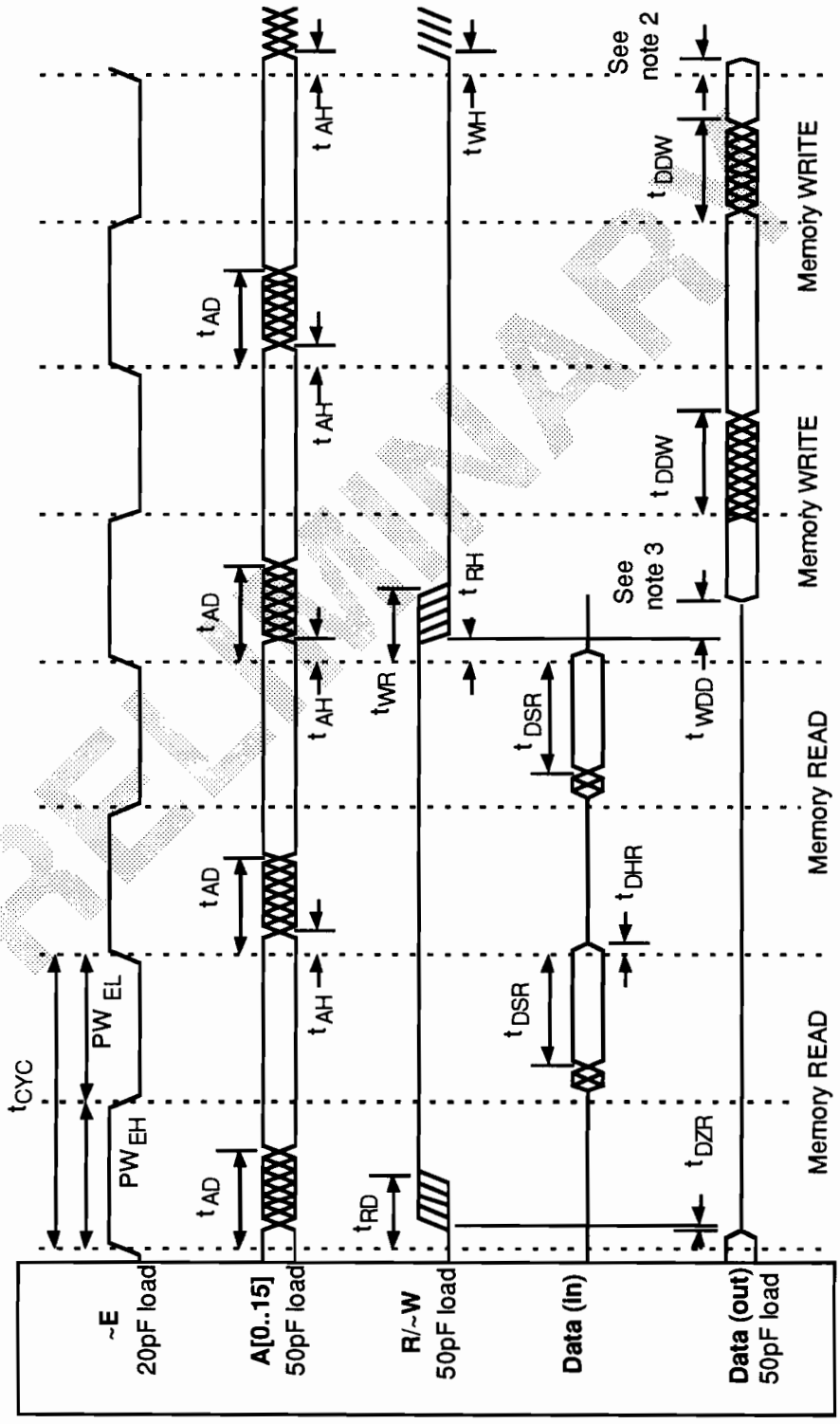


Figure 12.1 NEURON 3150 CHIP memory interface timing diagram

Appendix A: NEURON CHIP Configuration Data Structures

The software in the NEURON CHIP may be divided into three main sections: system image, application image, and network image

The System Image

This contains the LONTALK protocol, the NEURON C runtime library and the task scheduler. In the NEURON 3120 CHIP, this software is in the on-chip 10K ROM. In the NEURON 3150 CHIP, this software is in an external ROM. For the NEURON 3150 CHIP, this software is provided as part of the LONBUILDER Developer's Workbench environment. With this tool, the user can produce Intel Hex or Motorola S-record files containing the system image so that EPROM devices may be programmed.

In the NEURON 3120 CHIP, the following NEURON C runtime library functions are not in the on-chip ROM, but may be loaded into EEPROM along with the application program: `access_address`, `access_domain`, `access_nv`, `bcd2bin`, `bin2bcd`, `flush_wait`, `go_unconfigured`, `muldiv`, `muldivs`, `power_up`, `retrieve_status`, `reverse`, `update_address`, `update_domain`, `update_nv`, use of a signed bitfield, `memcpy` or structure assignment (length ≥ 256 bytes), `memset` (length ≥ 256 bytes), `memcpy` from `msg_in.data`, `memcpy` to `resp_out.data`, ability to send explicitly addressed messages, Neurowire slave mode.

The Application Image

This contains the object code generated by the NEURON C compiler from the user's application program, along with other application-specific parameters. These parameters may be queried by a network management tool. They include:

- Network variable fixed and self-identification data
- Program ID string
- Optional self-identification and self-documentation data
- Number of address table entries
- Number of domain table entries
- Number and size of network buffers
- Number and size of application buffers
- Number of receive transaction records
- Input clock speed of target NEURON CHIP
- Transceiver type and bit rate

In the NEURON 3150 CHIP, the application image is typically programmed into an external ROM, or downloaded over the network to EEPROM memory. In the NEURON 3120 CHIP, the application image is downloaded into the on-chip EEPROM memory. The LONBUILDER Developer's Workbench supports creation of application images.

The Network Image

This contains the address assignments of the LONWORKS node, the binding information connecting network variables and message tags between the nodes in the network, parameters of the LONTALK protocol that may be set at installation time, and configuration variables of the application program. A network management tool typically downloads the network image over the network into on-chip EEPROM memory when a node is installed. For simple networks, a node can update its own network image.

This section is intended for application programmers who need to understand the internal data structures of the NEURON CHIP that are used for address assignment, binding, and configuration. The NEURON C application program running on the NEURON

CHIP may access this data using run-time library calls and declarations for the purpose of node self-installation. In the LONBUILDER NEURON C development environment, function prototypes and declarations are found in the files `\LB\INCLUDE\ACCESS.H` and `\LB\INCLUDE\ADDRDEFS.H`.

These data structures may also be accessed by network management messages received over the network from a network management tool (see Appendix B). For network management tools running on host computers, the LONMANAGER application programmer's interface (API) provides convenient high-level access to these data structures from a host-based application program. See the *LONMANAGER API Programmer's Guide* for more details.

The NEURON CHIP configuration data structures may be divided into six main sections:

1. A fixed structure, whose size is independent of the application on the node.
2. A domain table, with an entry for every domain to which this node belongs.
3. An address table, with an entry for every network address referenced by this node.
4. Network variable tables, with entries for every network variable defined by this node.
5. Optional self-identification and self-documentation information describing this node and its network variables.
6. A channel configuration structure, defining the transceiver interface of the node.

A.1 Fixed Read-Only Data Structure

This structure is physically located at the start of on-chip EEPROM, at location 0xF000. It defines the node identification, as well as some of the application image parameters.

Declarations from `ACCESS.H` and `ADDRDEFS.H`

```
#define NEURON_ID_LEN 6
#define ID_STR_LEN    8

typedef struct {
      unsigned    neuron_id[ NEURON_ID_LEN ];
      unsigned    model_num;
      unsigned                        : 4;
      unsigned    minor_model_num     : 4;
const nv_fixed_struct * nv_fixed;
      unsigned    read_write_protect  : 1;
      unsigned                        : 1;
      unsigned    nv_count            : 6;
const snvt_struct * snvt;
      unsigned    id_string[ ID_STR_LEN ];
      unsigned                        : 1;
      unsigned    two_domains         : 1;
      unsigned                        : 6;
      unsigned    address_count       : 4;
} read_only_data_struct;

const read_only_data_struct read_only_data;
```

The application program may read, but not write this structure, using the global declaration `read_only_data`. The structure is 23 bytes long, and it may be read and mostly written (except for the first eight bytes) over the network using the Read Memory and Write Memory network management messages with `address_mode=1`. It is written during the process of downloading a new application image into the node.

A.1.1 Read-only Structure Field Descriptions

```
unsigned    neuron_id[ NEURON_ID_LEN ];   // offset 0x00
```

This field is a 6-byte ID assigned by the manufacturer of the NEURON CHIP which is unique to each NEURON CHIP manufactured. Hardware prevents this field from being written after manufacture. It may be read over the network using the Query ID network management message. It is also part of the unsolicited Service Pin network management message.

```
unsigned    model_num;                    // offset 0x06
unsigned    minor_model_num   : 4;        // offset 0x07
```

These are two fields that specify the model of the NEURON CHIP . The encoding of the model number field is: NEURON 3150 CHIP = 0 , NEURON 3120 CHIP = 8.

```
const nv_fixed_struct * nv_fixed;         // offset 0x08
```

This field is a pointer to the Network Variable fixed data table (see section A.4). If there are no network variables on the node, or this node is running the Microprocessor Interface Program, then this pointer is not useful.

```
unsigned    read_write_protect : 1;       // offset 0x0A
```

This bit specifies that parts of the NEURON CHIP memory may not be read or written over the network with the Read Memory and Write Memory network management messages (section B.1.5). The application program may set this bit with the NEURON C compiler directive `#pragma read_write_protect`. If this bit is set, only the Read-only Structure (section A.1), the SNVT Structures (section A.5), the Configuration Structure (section A.6), and the application data area may be read, and only the Configuration Structure may be written. The write-protected data includes the `read_write_protect` bit itself, so that once set, the bit may not be reset over the network.

```
unsigned    nv_count          : 6;
```

This field specifies the number of network variables declared in the application program running on this node (0-62). Each element of a network variable array is counted separately. If this node is running the Microprocessor Interface Program with the network variable configuration table on the host, this field is zero.

```
const snvt_struct * snvt;                 // offset 0x0B
```

This field is a pointer to the data structure that gives self-identification information for the network variables (see section A.5). If the self-identification information is not present, this is a null (0) pointer. If the NEURON CHIP is executing the Microprocessor Interface Program, this pointer is `0xFFFF`. The self-identification information may be suppressed with the NEURON C compiler directive `#pragma disable_snvt_si`.

```
unsigned    id_string[ ID_STR_LEN ];      // offset 0x0D
```

This field contains an 8-byte program identifying information as specified in either of the NEURON C compiler directives:

`#pragma set_id_string "`*`ssssssss`*`"`

or

`#pragma set_std_prog_id` *`fm:mm:mm:cc:cc:ss:ss:nn`*

The second format is reserved for Echelon-certified application nodes. The bit assignment for this format is as follows:

The first 4 bits are the format code (0x8 - 0xF).
The next 20 bits are the manufacturer code.
The next 16 bits are the device class.
The next 16 bits are the device sub-class.
The last 8 bits are the manufacturer-assigned model number.

The encoding of all these fields is TBD. If the program ID string is not specified by either of the compiler directives, then it contains the name of the NEURON C source file. The program ID string may be read over the network using the Query ID network management message. It is also part of the unsolicited Service Pin network management message.

```
unsigned    two_domains       : 1;        // offset 0x15
```

This bit specifies that the domain table has two entries (see section A.2). If this bit is zero, the domain table has one entry. This bit is set unless the NEURON C compiler directive `#pragma one_domain` was specified in the application program.

```
unsigned    address_count     : 4;        // offset 0x16
```

This field specifies the number of entries (0-15) in the address table (see section A.3).

A.2 The Domain Table

This table defines the domains to which this node belongs. It is located in EEPROM, and is part of the network image written during node installation. In the development environment, the contents of this table are downloaded when the node is loaded.

Declarations from `ACCESS.H` and `ADDRDEFS.H`

```
#define AUTH_KEY_LEN  6
#define DOMAIN_ID_LEN 6

typedef struct {
      unsigned    id[ DOMAIN_ID_LEN ];          // offset 0x00
      unsigned    subnet;                       // offset 0x06
      unsigned          : 1;
      unsigned    node  : 7;                    // offset 0x07
      unsigned    len;                          // offset 0x08
      unsigned    key[ AUTH_KEY_LEN ];          // offset 0x09
} domain_struct;

const domain_struct * access_domain( int index );
void update_domain( domain_struct * domain, int index );
```

The application program may read or write any entry in this table using the access routines `access_domain()` and `update_domain()`. The domain table consists of up to two entries, each 15 bytes in length. The default number of entries is two, which may be overridden by the NEURON C compiler directive `#pragma one_domain`. The entries in this table may be written and read over the network with the Update Domain, Leave Domain and Query Domain network management messages.

A.2.1 Domain Table Field Descriptions

```
unsigned    id[ DOMAIN_ID_LEN ];
```

Each domain in a LONWORKS network has a unique ID of 0, 1, 3 or 6 bytes in length. If the ID is shorter than 6 bytes, it is left justified in this field. In the development environment, the user specifies the size and value of the domain ID when creating the domain object.

```
unsigned    subnet;
```

This field specifies the ID of the subnet within this domain to which this node belongs. A subnet ID may be in the range 1 - 255 for each domain. In the development environment, this is assigned automatically when the subnet object is created. Zero is an invalid subnet ID.

`unsigned node;`

This field specifies the ID of the node within this subnet (1 – 127). In the development environment, this is assigned automatically when the node specification object is created. Zero is an invalid node ID.

`unsigned len;`

This field specifies the length of the domain ID in bytes (0, 1, 3 or 6). The value 0xFF (255) means that this domain table entry is not in use.

`unsigned key[ AUTH_KEY_LEN ];`

This field specifies the six-byte authentication key to be used in this domain for authenticated transactions. This key must match the key of all the other nodes on this domain that participate in authenticated transactions with this node. In the development environment, the user specifies the key in the node specification screen. The authentication key may be incremented over the network using the Update Key network management message.

A.3 The Address Table

This table defines the network addresses to which this node may send implicitly-addressed messages and network variables. It also defines the groups to which this node belongs. It is located in EEPROM, and is part of the network image written during node installation.

Declarations from `ACCESS.H` and `ADDRDEFS.H`

```
typedef enum { UNBOUND, SUBNET_NODE, NEURON_ID, BROADCAST } addr_type;

typedef union {
      group_struct      gp;
      snode_struct      sn;
      bcast_struct      bc;
      turnaround_struct ta;
} address_struct;

const address_struct * access_address( int index );
update_address(  const address_struct * address_entry, int index );
int addr_table_index( message_tag_name );
```

The application program may read and write any entry in this table using the access routines `access_address()` and `update_address()`. The index of the address table entry corresponding to any message tag declared in the program may be determined with the function `addr_table_index()`. The address table consists of up to 15 entries, each 5 bytes in length. The default number of entries is 15, which may be overridden by the NEURON C compiler directive `#pragma num_addr_table_entries` *nn*.

Each entry may be in one of five formats: group address, subnet/node address, broadcast address, turnaround address, or not in use. A group address is used for multicast addressing, when a network variable or message tag is used in a connection having more than two members. A subnet/node address is used for unicast addressing, when an output network variable or message tag is used in a connection with one other node. Broadcast addressing is not used by the LONBUILDER binder or the LONMANAGER API binder when they create network variable or message tag connections. However, broadcast addressing is supported in the protocol, and may be used with explicit addressing. User software that implements installation may also install broadcast address table entries in a node . A turnaround address is used for network variables that are only bound to other network variables in the same node, and not to any network variables on other nodes. Destination addresses in the unique 48-bit NEURON ID format are never used in the address table, but they may be used as destination addresses in explicitly addressed messages. For

completeness, this format is described below. The declaration of this format is in the NEURON C include file `MSG_ADDR.H`.

In the development environment, entries in the address table are created when the network manager loads the network image into the node. The entries in this table may be read and written over the network with the Query Address and Update Address Table network management messages. An entry using group address format may be updated over the network with the Update Group Address Data network management message.

The first byte in an address table entry specifies the format of the entry:

0	not in use / turnaround format
1	(subnet,node) format
3	broadcast format
128 - 255	group format

Any bindable message tags declared in the application program are assigned to the first entries in the address table in order of declaration. This is followed by address table entries used for network variables – in the development environment, the binder assigns these entries.

The repeat timer, retry count, receive timer and transmit timer are common to several of these address formats, and are described below in section A.3.11.

A.3.1 Declaration of Group Address Format

```
typedef struct {
      unsigned    type        : 1;        // offset 0x00
      unsigned    size        : 7;
      unsigned    domain      : 1;        // offset 0x01
      unsigned    member      : 7;
      unsigned    rpt_timer   : 4;        // offset 0x02
      unsigned    retry       : 4;
      unsigned    rcv_timer   : 4;        // offset 0x03
      unsigned    tx_timer    : 4;
      unsigned    group       : 8;        // offset 0x04
} group_struct;
```

A.3.2 Group Address Field Descriptions

`unsigned  type  : 1;`

This bit is 1 for a group address, 0 for any of the other formats.

`unsigned  size  : 7;`

This field specifies the size of the group (2-64). The size of a group includes the sender of the message. If this field is 0, then the group is of unlimited size, and Unacknowledged or Unacknowledged/Repeated service must be used.

`unsigned  domain  : 1;`

This field specifies the index into the domain table for this address (0 or 1).

`unsigned  member  : 7;`

This field specifies the member ID of this node within this group (0-63). A group of unlimited size has 0 in this field. The member ID is used in acknowledgments to allow the sender of an acknowledged multicast message to keep track of which nodes have responded.

`unsigned  group  : 8;`

This field specifies the ID of this group within this domain. A group ID may be in the range of 0 - 255. In the development environment, the group ID is allocated by the binder.

A.3.3 Declaration of Subnet/Node Address Format

```
typedef struct {
      addr_type   type;                    // offset 0x00
      unsigned    domain       : 1;        // offset 0x01
      unsigned    node         : 7;
      unsigned    rpt_timer    : 4;        // offset 0x02
      unsigned    retry        : 4;
      unsigned                 : 4;        // offset 0x03
      unsigned    tx_timer     : 4;
      unsigned    subnet       : 8;        // offset 0x04
} snode_struct;
```

A.3.4 Subnet/Node Address Field Descriptions

`addr_type type;`

This field contains the value `SUBNET_NODE` (1).

`unsigned domain : 1;`

This field specifies the index into the domain table for this address (0 or 1).

`unsigned node : 7;`

This field specifies the node ID (1-127) within the specified subnet and domain. Zero is not a valid node ID.

`unsigned subnet : 8;`

This field specifies the subnet ID (1-255) within the specified domain. Zero is not a valid subnet ID.

A.3.5 Declaration of Broadcast Address Format

```
typedef struct {
      addr_type   type;                    // offset 0x00
      unsigned    domain       : 1;        // offset 0x01
      unsigned                 : 1;
      unsigned    backlog      : 6;
      unsigned    rpt_timer    : 4;        // offset 0x02
      unsigned    retry        : 4;
      unsigned                 : 4;        // offset 0x03
      unsigned    tx_timer     : 4;
      unsigned    subnet       : 8;        // offset 0x04
} bcast_struct;
```

A.3.6 Broadcast Address Field Descriptions

`addr_type type;`

This field contains the value `BROADCAST` (3).

`unsigned domain : 1;`

This field specifies the index into the domain table for this address (0 or 1).

`unsigned backlog : 6;`

This field specifies an estimate of the channel backlog that would be created by an acknowledged or request/response message broadcast using this address. It should be set to the expected number of acknowledgements or responses. If this is unknown, this field is zero.

`unsigned subnet : 8;`

This field specifies the subnet number (1-255) within the specified domain. The message is delivered to all nodes in this subnet. If the subnet number is 0, the message

is delivered to all nodes in the domain. If Request/Response or Acknowledged service is used with a broadcast address, the transaction completes as soon as the first response or acknowledgment is received. Subsequent responses or acknowledgments are discarded.

A.3.7 Declaration of Turnaround Address Format

```
typedef struct {
      addr_type   type;                    // offset 0x00
      unsigned    turnaround;              // offset 0x01
      unsigned    rpt_timer    : 4;        // offset 0x02
      unsigned    retry        : 4;
      unsigned                 : 4;        // offset 0x03
      unsigned    tx_timer     : 4;
} turnaround_struct;
```

A.3.8 Turnaround Address Field Descriptions

`addr_type type;`

Contains the value UNBOUND (0)

`unsigned turnaround;`

This field contains the value one. If the turnaround field is zero, this address table entry is not in use.

A.3.9 Declaration of NEURON ID Address Format

Note: this format is not used in the address table, but it may be used as a destination address for an explicitly addressed message.

```
typedef struct {
      addr_type   type;                    // offset 0x00
      unsigned    domain       : 1;        // offset 0x01
      unsigned                 : 7;
      unsigned    rpt_timer    : 4;        // offset 0x02
      unsigned    retry        : 4;
      unsigned                 : 4;        // offset 0x03
      unsigned    tx_timer     : 4;
      unsigned    subnet       : 8;        // offset 0x04
      unsigned    nid[ NEURON_ID_LEN ];    // offset 0x05
} nrnid_struct;
```

A.3.10 Neuron ID Address Field Descriptions

`addr_type type;`

This field contains the value NEURON_ID (2).

`unsigned domain : 1;`

This field specifies the index into the domain table for the destination address (0 or 1). However, unique-ID addressed messages may be sent on any domain. Unconfigured nodes will receive messages addressed in unique-ID format on any domain. When this occurs, the message is said to be received on the flexible domain, and any response or acknowledgement will be sent back on the domain it was received in, with source subnet and node identifiers of zero.

`unsigned subnet : 8;`

This field specifies the destination subnet number (1-255) within the domain. It is only used for routing of the message, and may be set to zero if the message should pass through all routers in the domain.

`unsigned nid[ NEURON_ID_LEN ];`

This field specifies the unique 48-bit ID of the destination NEURON CHIP.

A.3.11 Timer Field Descriptions

In the development environment, the user specifies these values in the network variable or message connection screens.

```
unsigned    rpt_timer    : 4;
```

This field specifies the time interval between repetitions of an outgoing message when Unacknowledged/Repeated service is used. The encoding of this field is in table A.1.

```
unsigned    retry        : 4;
```

This field specifies the number of retries for Acknowledged, Request/Response, or Unacknowledged/Repeated service (0-15). The maximum number of messages sent is one more than this number.

```
unsigned    rcv_timer    : 4;
```

When the node receives a multicast (group) message, the receive timer is set to the time interval specified by this field. If a message with the same transaction ID is received before the receive timer expires, it is considered to be a retry of the previous message. The encoding of this field is in table A.1.

```
unsigned    tx_timer     : 4;
```

This field specifies the time interval between retries when Acknowledged or Request/Response service is used. The transaction retry timer is restarted when each attempt is made, and also when any acknowledgment or response (except for the last one) is received. For Request/Response service, the requesting node should take into account the delay necessary for the application to respond when setting the transaction timer. This is important, for example, for network management messages that write into EEPROM. The encoding of the transaction timer field is specified in table A.1.

Value	rpt_timer	rcv_timer	tx_timer	non_group_timer
0	16	128	16	128
1	24	192	24	192
2	32	256	32	256
3	48	384	48	384
4	64	512	64	512
5	96	768	96	768
6	128	1,024	128	1,024
7	192	1,536	192	1,536
8	256	2,048	256	2,048
9	384	3,072	384	3,072
10	512	4,096	512	4,096
11	768	6,144	768	6,144
12	1,024	8,192	1,024	8,192
13	1,536	12,288	1,536	12,288
14	2,048	16,384	2,048	16,384
15	3,072	24,576	3,072	24,576

Table A.1 Encoding of timer field values in msec.

A.4 Network Variable Tables

There are two tables associated with network variables: the network variable configuration table and the network variable fixed table. The network variable configuration table defines the configurable attributes of the network variables in this node. It is located in EEPROM so that it can be modified during node installation and is part of the network image written during node installation. The network variable fixed

table defines the compile and link-time attributes of the network variables in this node. It may be located in read-only memory and is part of the application image written during node manufacture.

For a node running the Microprocessor Interface Program, the network variable fixed table, and the network variables themselves are located in the memory of the host microprocessor. The network variable configuration table may either be in the memory of the NEURON CHIP, or in the memory of the host microprocessor. In the latter case, the limit of 62 network variables per node does not apply, and the LONTALK protocol firmware on the NEURON CHIP does not process network variable update messages, but instead passes them to the host microprocessor.

Declarations from ACCESS.H

```
#define MAX_NVS 62

typedef struct {
      unsigned    nv_priority       : 1;          // offset 0x00
      unsigned    nv_direction      : 1;
      unsigned    nv_selector_hi    : 6;
      unsigned    nv_selector_lo    : 8;          // offset 0x01
      unsigned    nv_turnaround     : 1;          // offset 0x02
      unsigned    nv_service        : 2;
      unsigned    nv_auth           : 1;
      unsigned    nv_addr_index     : 4;
} nv_struct;

const nv_struct * access_nv( int index );
void update_nv( const nv_struct * nv_entry, int index );
int nv_table_index( network_variable_name );

typedef struct {
      unsigned    nv_sync     : 1;              // offset 0x00
      unsigned                : 2;
      unsigned    nv_length   : 5;
      void        * nv_address;                 // offset 0x01
} nv_fixed_struct;
```

The application program may read and write any entry in the network variable configuration table using the access routines `access_nv()` and `update_nv()`. The index of the table entry corresponding to any network variable declared in the program may be determined with the function `nv_table_index()`. The base address of the network variable fixed table may be retrieved from the read-only data field `read_only_data.nv_fixed`.

Each table consists of up to 62 entries, each 3 bytes in length. The number of entries is determined by the number of network variables declared in the application program – each element of a network variable array counts separately. The index into either of these tables corresponding to a particular network variable is determined by the order of declaration of the network variables in the application program. Entries in the network variable configuration table may be read and written over the network with the Query/Update Net Variable Configuration network management messages. The network variable fixed table may not be written, except when downloading the application image.

A.4.1 Network Variable Configuration Table Field Descriptions

```
unsigned    nv_priority       : 1;
```

This bit is set to 1 if the network variable uses priority messaging. Specified by `bind_info( priority | nonpriority )` in the NEURON C declaration of the network variable. In the development environment, this may be overridden in the connection screen.

```
unsigned    nv_direction       : 1;
```

This bit is set to one if this is an output network variable, 0 if an input. This bit must not be changed by the application program or an Update Net Variable Configuration network management message.

```
unsigned    nv_selector_hi     : 6;
unsigned    nv_selector_lo     : 8;
```

These two fields form a 14-bit network variable selector in the range 0 - 0x3FFF. Selector values 0x3000 - 0x3FFF are reserved for unbound network variables, with the selector value equal to 0x3FFF minus the network variable index. Selector values 0 - 0x2FFF are available for bound network variables. The input network variables on any one node must all have different selectors, unless they are turnaround network variables bound to each other. The binder in the development system ensures this by assigning a network variable selector to the union of all the connection sets that this network variable participates in. This, however, is not required by the protocol.

```
unsigned    nv_turnaround      : 1;
```

This bit is set to one if this is a turnaround network variable, that is, bound to another network variable on the same node.

```
unsigned    nv_service         : 2;
```

This field specifies the type of service used to deliver this network variable. Specified by `bind_info( ackd | unackd_rpt | unackd )` in the NEURON C declaration of the network variable. In the development environment, this may be overridden in the connection screen. The encoding is as follows:

```
ACKD        = 0   Acknowledged
UNACKD_RPT  = 1   Unacknowledged/Repeated
UNACKD      = 2   Unacknowledged
```

```
unsigned    nv_auth            : 1;
```

This bit is set to one if this network variable uses authenticated transactions. Specified by `bind_info( authenticated | nonauthenticated )` in the NEURON C declaration of the network variable. In the development environment, this may be overridden in the connection screen.

```
unsigned    nv_addr_index      : 4;
```

This field specifies the index into the address table for this network variable (0-14). If the network variable is not associated with an address table entry, then the value 15 is used.

A.4.2 Network Variable Fixed Table Field Descriptions

```
unsigned    nv_sync            : 1;
```

This bit is set to one if this is a synchronous network variable. Specified with the modifier `sync` in the NEURON C declaration of the network variable.

```
unsigned    nv_length          : 5;
```

This field specifies the number of bytes in the network variable (1-31).

```
void        * nv_address;
```

This field is a pointer to the location of the network variable's data in RAM or EEPROM.

A.5 The Standard Network Variable Type (SNVT) Structures

There are four structures associated with self-identification and self-documentation: a fixed size SNVT structure, a table containing a self-identification descriptor for each network variable, a self-documentation string for the node, and self-documentation information for network variables. If the NEURON C compiler directive `#pragma disable_snvt_si` is specified in the application program, none of this information is present. These tables may be located in read-only memory, and form part of the application image written during node manufacture.

Declarations from `ACCESS.H`

```
typedef struct {
      unsigned long     length;              // offset 0x00
      unsigned          num_netvars;         // offset 0x02
      unsigned          version;             // offset 0x03
      unsigned          mtag_count;          // offset 0x04
} snvt_struct;

typedef struct {
      unsigned    ext_rec      : 1;          // offset 0x00
      unsigned    nv_sync      : 1;
      unsigned    nv_polled    : 1;
      unsigned    nv_offline   : 1;
      unsigned    nv_service_type_config  : 1;
      unsigned    nv_priority_config      : 1;
      unsigned    nv_auth_config          : 1;
      unsigned    nv_config_class         : 1;
      unsigned    snvt_type_index;
} snvt_desc_struct;
```

For a node running a standard application program, the base address of the SNVT Structure may be retrieved from the read-only data field `read_only_data.snvt`. For a node running the Microprocessor Interface Program, the SNVT Structures are located in the memory of the host microprocessor and may be read with the Query SNVT network management message (the maximum size of the SNVT structure is 65,536 bytes). The SNVT header structure is five bytes long. The SNVT descriptor table follows immediately afterwards, and it has one entry for every network variable declared in the application program. Each element of a network variable array has its own entry. Each entry in the SNVT descriptor table is two bytes long.

Following the SNVT descriptor table is a null-terminated text string containing the self-documentation of the node. This string may be up to 255 bytes in length, and it is the string specified in the NEURON C compiler directive `#pragma set_node_sd_string "sss"`. If there is no self-documentation string, the null termination byte is still present.

Following the self-documentation string for the node are extension records for those network variables that require them. All the self-identification and self-documentation information may be read over the network with the Read Memory network management command using `address_mode=0`.

A.5.1 SNVT Structure Field Descriptions

`unsigned long length;`

This field specifies the total number of bytes in the self-identification and self-documentation data structures described here.

```
unsigned    num_net_vars;
```

This field specifies the number of network variables declared in this node. Each element of a network variable array counts separately.

```
unsigned    version;
```

This field specifies the format of the SNVT structures to follow. The format described in this section is version zero.

```
unsigned    mtag_count;
```

This field specifies the number of bindable message tags declared by the application program on this node. These message tags take the first entries in the address table (see section A.3).

A.5.2 SNVT Descriptor Table Field Descriptions

```
unsigned    ext_rec     : 1;
```

This bit is set to one if this network variable has an extension record following the node's self-documentation string.

```
unsigned    nv_sync     : 1;
```

This bit is set to one if this is a synchronous network variable. Specified with the modifier `sync` in the NEURON C declaration of the network variable.

```
unsigned    nv_polled   : 1;
```

This bit is set to one if this network variable is polled. If it is an output network variable, this is specified with the modifier `polled` in the NEURON C declaration. If this is an input network variable, this means that the network variable is mentioned as the argument of a qualified `poll()` system call, or that there is an unqualified `poll()` call in the application program.

```
unsigned    nv_offline  : 1;
```

This bit is set to one if a network management tool should take the node off-line before this network variable is updated. Specified by `bind_info( offline )` in the NEURON C declaration of the network variable.

```
unsigned    nv_service_type_config  : 1;
```

This bit is set to one if the service type of this network variable (`ACKD`, `UNACKD`, `UNACKD_RPT`) may be modified by a network management message. Specified by `bind_info( service_type( config | nonconfig ) )` in the NEURON C declaration of the network variable.

```
unsigned    nv_priority_config      : 1;
```

This bit is set to one if the priority of this network variable may be modified by a network management message. Specified by `bind_info( priority | nonpriority ( config | nonconfig ) )` in the NEURON C declaration of the network variable.

```
unsigned    nv_auth_config          : 1;
```

This bit is set to one if the authentication of this network variable may be modified by a network management message. Specified by `bind_info( auth | nonauth ( config | nonconfig ) )` in the NEURON C declaration of the network variable.

```
unsigned    nv_config_class         : 1;
```

This bit is set to one if this is a configuration network variable whose value is stored in EEPROM. Specified with the `config` modifier in the NEURON C declaration of the network variable.

```
unsigned    snvt_type_index;
```

If this is a network variable of a standard type, this field specifies the type (1-250). For a listing of the Standard Network Variable Types, see *The SNVT Guide.* If this field is zero, the network variable is of a non-standard type.

A.5.3 SNVT Table Extension Records

Extension records may be present for only some of the network variables. If an extension record is present, the field `ext_rec` is set to one in the SNVT descriptor. The extension records appear in the order that the network variables were declared in the application program. Each extension record begins with a one-byte bit-mask defining which fields are to follow. The fields follow in the order of the bits in the bit mask defined here. These bits appear in the mask starting with the most significant bit.

```
unsigned    mre : 1;
```

If this bit is set to one, the extension record contains an estimate of the maximum rate at which this network variable is updated. The field is an `unsigned int` in the range 0-127 representing the rate in the range 0 - 1878.0 as specified by `bind_info( max_rate_est( nnn ) )` in the NEURON C declaration of the network variable. The rate is given by the formula $2^{(n/8)-5}$ messages/per second, rounded to the nearest tenth of a message per second.

```
unsigned    re : 1;
```

If this bit is set to one, the extension record contains an estimate of the average rate at which this network variable is updated. The field is an `unsigned int` in the range 0-127 representing the rate in the range 0 - 1878.0 as specified by `bind_info( rate_est( nnn ) )` in the NEURON C declaration of the network variable. The rate is given by the formula $2^{(n/8)-5}$ messages/per second, rounded to the nearest tenth of a message per second.

```
unsigned    nm : 1;
```

If this bit is set to one, the extension record contains the name of the network variable as declared in the NEURON C program. This information is present if the compiler directive `#pragma enable_sd_nv_names` is specified. The name is represented as a null terminated string of up to 17 bytes, or up to 21 bytes if it is a network variable array element. Each array element has its own extension record containing the name of the array, a left square bracket, one or two decimal digits denoting the index of the element, and a right square bracket.

```
unsigned    sd : 1;
```

If this bit is set to one, the extension record contains the self-documentation string for the network variable. This is a null-terminated string of up to 1023 bytes, which may be specified by the modifier `sd_string( "sss" )` in the NEURON C declaration of the network variable.

A.6 The Configuration Structure

This structure defines the hardware and transceiver properties of this node. It is located in EEPROM, and parts of it belong to the application image written during node manufacture, and to the network image written during node installation.

Declarations from `ACCESS.H`

```
#define LOCATION_LEN  6
#define NUM_COMM_PARAMS 7

typedef struct {
      unsigned long     channel_id;                    // offset 0x00
      char              location[ LOCATION_LEN ];      // offset 0x02
      unsigned          comm_clock        : 5;         // offset 0x08
      unsigned          input_clock       : 3;
      unsigned          comm_type         : 3;         // offset 0x09
      unsigned          comm_pin_dir      : 5;
```

```
      unsigned          reserved[ 5 ];                  // offset 0x0A
      unsigned          node_priority;                  // offset 0x0F
      unsigned          channel_priorities;             // offset 0x10
      union {
          unsigned      xcvr_params[ NUM_COMM_PARAMS ];
          direct_param_struct dir_params;               // offset 0x11
      }
      unsigned          non_group_timer    : 4;         // offset 0x18
      unsigned          nm_auth            : 1;
      unsigned          preemption_timeout : 3;
} config_data_struct;

typedef struct {
      unsigned          collision_detect   : 1;         // offset 0x11
      unsigned          bit_sync_threshold : 2;
      unsigned          filter             : 2;
      unsigned          hysteresis         : 3;
      unsigned                             : 6;         // offset 0x12
      unsigned          cd_tail            : 1;
      unsigned          cd_preamble        : 1;
} direct_param_struct;

const config_data_struct config_data;
```

The application program may read, but not write this structure using the global declaration `config_data`. The structure is 25 bytes long, and it may be read and written over the network using the Read Memory and Write Memory network management messages with `address_mode=2`. The Media Access Control processor reads the channel parameters (offsets 0x08 through 0x17) from the configuration structure when the node is reset and initializes the transceiver. Therefore, after changing any of the channel parameters, the node should be reset for the changes to take effect.

A.6.1 Configuration Structure Field Descriptions

```
unsigned long     channel_id;
```

This field specifies the ID of the channel that this node is assigned to. In the development environment, this is assigned automatically when the channel object is created. The NEURON CHIP firmware does not reference this field, but it is available to assist in recreation of the network topology data structure by interrogating the nodes.

```
char              location[ LOCATION_LEN ];
```

The location field is used to pass a 6-byte ASCII string describing the physical location of the node to the network management node. In the development environment, this is defined in the node specification screen.

```
unsigned          comm_clock        : 5;
```

For direct mode transceivers, this field specifies the ratio between the NEURON CHIP input clock oscillator frequency and the transceiver bit rate. For special purpose mode transceivers, it specifies the rate of the bit clock between the NEURON CHIP and the transceiver. In the development environment, the transceiver bit rate is specified in the channel screen. Table A.2 shows the transceiver bit rate as a function of the `input_clock` field and the `comm_clock` field.

		input_clock				
		5	4	3	2	1
comm_clock	ratio	10 MHz	5 MHz	2.5 MHz	1.25 MHz	625 kHz
0	8:1	1,250	625	312.5	156.3	78.1
1	16:1	625	312.5	156.3	78.1	39.1
2	32:1	312.5	156.3	78.1	39.1	19.5
3	64:1	156.3	78.1	39.1	19.5	9.8
4	128:1	78.1	39.1	19.5	9.8	4.9
5	256:1	39.1	19.5	9.8	4.9	—
6	512:1	19.5	9.8	4.9	—	—
7	1,024:1	9.8	4.9	—	—	—

Table A.2 Transceiver bit rate (kbit/s) as a function of `comm_clock` and `input_clock`

```
unsigned        input_clock     : 3;
```

This field specifies the NEURON CHIP input clock (oscillator frequency). In the development environment, the user specifies this value in the hardware properties screen for the node. The encoding is as follows:

5	10.0 MHz
4	5.0 MHz
3	2.5 MHz
2	1.25 MHz
1	625 kHz
0	not used

```
unsigned        comm_type       : 3;
```

This field specifies the type of transceiver. In the development environment, this is specified in the channel screen. The encoding is as follows:

0	Blank NEURON CHIP
1	Single-ended
5	Differential
2	Special-purpose

```
unsigned        comm_pin_dir    : 5;
```

This field specifies the direction of the NEURON CHIP's communications port pins. Zero indicates an input, one indicates an output with respect to the NEURON CHIP. The least significant bit corresponds to pin CP0. Values used in this field include:

0x00	Blank NEURON CHIP
0x0E	Direct mode - single-ended
0x0C	Direct mode - differential
0x1E	Special-purpose - wake-up pin is an output
0x17	Special-purpose - wake-up pin is an input

```
unsigned        reserved[ 5 ];
```

This field specifies the raw transceiver parameters in units of CPU cycles. In the development environment, these parameters are specified as the Raw Data in the channel screen. The values in these fields are the repetition counts for software delay loops in the firmware that control the timing of the media access algorithm. See Figure 6.3 for a description of the timing of packet transmission. The fields are the following:

`reserved[ 0 ]`	preamble length pad – determines the length of the preamble for direct mode transceivers
`reserved[ 1 ]`	packet cycle pad – determines the packet cycle duration for counting down the backlog
`reserved[ 2 ]`	beta pad - the pad for the beta loop
`reserved[ 3 ]`	transmit beta 1 pad – the pad for the beta 1 period after transmitting
`reserved[ 4 ]`	receive beta 1 pad – the pad for the beta 1 period after receiving

```
unsigned          node_priority;
```

This field specifies the priority slot used by the node when sending priority messages on the channel (1-255). It should not be greater than the number of priority slots on the channel. If the node has no priority slot allocated, this is zero. In the development environment, the node priority is specified in the hardware properties screen for the node.

```
unsigned          channel_priorities;
```

This field specifies the number of priority slots on the channel (0-255). The slots are numbered starting at one. In the development environment, this is specified in the channel screen, with a maximum value of 127.

```
unsigned          xcvr_params[ NUM_COMM_PARAMS ];
```

This field forms an array of seven transceiver-specific parameters for special-purpose mode transceivers. All seven parameters are loaded into the transceiver when the node is initialized. In the development environment, these are specified as General Purpose Data in the channel screen. The most significant bit of the first transceiver parameter is defined to be the alternate channel bit. The alternate channel is used for the last two transmission attempts when using Acknowledged, Request/Response or Unacknowledged/Repeated service. All other transceiver parameters are user-defined. For direct-mode transceivers, the first two bytes are overlaid by the direct mode parameter structure defined below.

```
unsigned          non_group_timer    : 4;
```

When the node receives a unicast or broadcast (non-group) message requiring a response or acknowledgment, the receive timer is set to the time interval specified by this field. If a message with the same transaction ID is received before the receive timer expires, it is considered to be a retry of the previous message. In the development environment, this is implicitly set to the maximum of the non-group receive timers specified in the connection screens. The encoding is of this field is in table A.1. When network management messages are received by an unconfigured node with unique ID addressing, a receive timer of 8.192 seconds is used, rather than the value of this field.

```
unsigned          nm_auth : 1;
```

This field specifies that network management messages are to be authenticated. Setting this bit prevents the node from being configured by an unauthorized network management node. However, network management messages received on the flexible domain (when the node is unconfigured) cannot be authenticated. In the development environment, network management authentication is defined in the node specification screen.

```
unsigned          preemption_timeout       : 3;
```

This field specifies the maximum time the node should wait for a free buffer. In the development environment, this is defined in the node specification screen. The encoding is as follows:

0	forever
1	2 secs
2	4 secs
3	6 secs

4	8 secs
5	10 secs
6	12 secs
7	14 secs

A.6.2 Direct Mode Transceiver Parameters Field Descriptions

For direct (single-ended or differential) mode transceivers, these parameters are used to control the operation of the transceiver port. In the development environment, these parameters are specified in the channel screen.

```
unsigned          collision_detect  : 1;
```

This field specifies that the NEURON CHIP monitors pin CP4 for an indication of a collision on the network.

```
unsigned          bit_sync_threshold: 2;
```

This field specifies the number of logic one bits received that are to be interpreted as the bit sync indicating the start of a packet. The encoding is as follows:

Threshold	Number of bits
0	4
1	5
2	6
3	7

```
unsigned          filter            : 2;
```

For differential mode transceivers, this field specifies the setting of the receive glitch filter (0-3). See the electrical specifications of the NEURON CHIP (table 12.6) for details of the available glitch filter settings.

```
unsigned          hysteresis        : 3;
```

For differential mode transceivers, this field specifies the setting of the receive hysteresis filter (0-7). See the electrical specifications of the NEURON CHIP (table 12.5) for details of the available hysteresis filter settings.

```
unsigned          cd_tail           : 1;
```

This bit specifies that collisions are to be detected at the end of the transmitted packet following the code violation.

```
unsigned          cd_preamble       : 1;
```

This bit specifies that collisions are to be detected during the preamble at the beginning of the transmitted packet. When specified, the packet is terminated at the end of the preamble if a collision is detected during the preamble.

Appendix B: Network Management and Diagnostic Services

In addition to application message services, the LONTALK protocol provides network management services for installation and configuration of nodes, downloading of software, and diagnosis of the network. Message codes used by the LONTALK protocol are as follows:

Message Type	Hexadecimal Message Codes
Application Messages	0x00 - 0x3E
Application Responses	0x00 - 0x3E
Response if node is off-line	0x3F

Foreign Messages	0x40 - 0x4E
Foreign Responses	0x40 - 0x4E
Response if node is off-line	0x4F

Section 7.3 describes the use of application messages. A foreign message is a packet of another protocol embedded in the LONTALK protocol packet. The application program allocates application or foreign message and response codes.

Network Diagnostic Messages	Request Code	Success Response	Failed Response
Query Status	0x51	0x31	0x11
Proxy Command	0x52	0x32	0x12
Clear Status	0x53	0x33	0x13
Query XCVR Status	0x54	0x34	0x14

Table B.1 Network diagnostic messages

Network Management Messages	Request Code	Success Response	Failed Response
Query ID	0x61	0x21	0x01
Respond to Query	0x62	0x22	0x02
Update Domain	0x63	0x23	0x03
Leave Domain	0x64	0x24	0x04
Update Key	0x65	0x25	0x05
Update Address	0x66	0x26	0x06
Query Address	0x67	0x27	0x07
Query Net Variable Config	0x68	0x28	0x08
Update Group Address Data	0x69	0x29	0x09
Query Domain	0x6A	0x2A	0x0A
Update Net Variable Config	0x6B	0x2B	0x0B
Set Node Mode	0x6C	0x2C	0x0C
Read Memory	0x6D	0x2D	0x0D
Write Memory	0x6E	0x2E	0x0E
Checksum Recalculate	0x6F	0x2F	0x0F
Wink	0x70	–	–
Memory Refresh	0x71	0x31	0x11
Query SNVT	0x72	0x32	0x12
Network Variable Fetch	0x73	0x33	0x13

Table B.2 Network management messages

Router Configuration Messages	0x74 - 0x7E
Router Config Success Responses	0x34 - 0x3E
Router Config Failed Responses	0x14 - 0x1E

Router messages are used by network management tools to configure nodes that run the special router system image. They are not useful for application nodes, which will return the failed response.

Service Pin Message	0x7F	(Unsolicited message)
Network Variable Messages	0x80 - 0xFF	

Network variable messages cannot be received by an application program using explicit messaging syntax. They are sent by updating output network variables in the application program, and are implicitly received by input network variables in the same connection. For a discussion of network variable messages, see section B.3.

Network management and network diagnostic messages may be delivered using Request/Response service (except for *mode on-line, mode off-line* and *wink*, which do not have response data associated with them). Network management messages that do not have response data associated with them may also be delivered with the other classes of service, namely Acknowledged, Unacknowledged and Unacknowledged/Repeated. These messages are: Respond to Query, Update Domain, Leave Domain, Update Key, Update Address, Update Group Address Data, Update Net Variable Config, Set Node Mode, Write Memory, Checksum Recalculate, Memory Refresh and Clear Status. If broadcast addressing is used with Acknowledged or Request/Response service, then only the first acknowledgement or response can be handled.

Application messages and network variable updates are delivered with the specified class of service. Note that most network management and network diagnostic messages may be authenticated (if the `nm_auth` bit is set in the Configuration Structure). Authentication never applies to Query ID, Respond to Query, Query Status, or Proxy messages.

For NEURON C programs, these message structures are conveniently defined in the include file `NETMGMT.H`

In the following descriptions of the network management messages, the data structures named `NM_xxx_request` specify the data field of the outgoing message (following the code field). Similarly, the data structures named `NM_xxx_response` specify the data field of the corresponding response. Network management messages are sent like any other explicit message, either from a host microprocessor or from a NEURON CHIP. The following example shows how a NEURON C program running on a NEURON CHIP could use the Read Memory network management message (section B.1.5) to retrieve the 6-byte NEURON ID from another NEURON CHIP at offset 0x0000 into the Read-only Structure:

```
#include <NETMGMT.H>
NM_read_memory_request read_rq;         // a local copy of the request msg

msg_tag read_mem_tag;                   // declare a destination address

when( reset ) {
    msg_out.code = 0x6D;                // Read-memory code
    read_rq.mode = READ_ONLY_RELATIVE; // Address mode
    read_rq.offset = 0x0000;            // Address offset
    read_rq.count = 6;                  // Byte count
    memcpy( msg_out.data, &read_rq, sizeof( read_rq ) );
                                        // Copy into msg_out data array
    msg_out.service = REQUEST;          // Expect a response
    msg_out.tag = read_mem_tag;         // Destination address
    msg_send( );                        // Send the message
}

unsigned neuron_id[ 6 ];                // Place to save the returned ID

when( resp_arrives( read_mem_tag ) ) {
    memcpy( neuron_id, resp_in.data, 6 );
        // copy the response data to a local variable
}
```

The failed response is returned when the destination NEURON CHIP cannot process the message. For example, when a table index is out of range, there is a memory failure when writing to EEPROM, or an attempt is made to violate read/write protection.

B.1 Network Management Messages

These messages are described in six main groups: node identification messages, domain table messages, address table messages, messages related to network variables, node memory messages and special-purpose messages.

B.1.1 Node Identification Messages

Query ID (Request/Response only)

This message requests selected nodes to respond with a message containing their 48-bit NEURON ID and program ID. This message is normally broadcast during network installation to find specific nodes in the domain. It can be used to find unconfigured nodes or explicitly selected nodes, or nodes with specified memory contents at a specified address. This address may be specified absolutely, relative to the Read-only Structure (section A.1), or relative to the Configuration Structure (section A.6). Only data within the Read-only Structure , the SNVT Structures (section A.5) or the Configuration Structure, may be matched. This can be used for example to find nodes with a particular channel ID or location string (in the Configuration Structure) or program ID string (in the Read-only Structure).

Message declarations from NETMGMT.H:

```
typedef struct {
    enum {
        UNCONFIGURED     = 0,
        SELECTED         = 1,
        SELECTED_UNCNFG  = 2,
    } selector;
    enum {                      // Begin optional fields
        ABSOLUTE            = 0,
        READ_ONLY_RELATIVE  = 1,
        CONFIG_RELATIVE     = 2,
    } mode;
    unsigned long offset;
    unsigned count;
    unsigned data[];
} NM_query_id_request;

typedef struct {
    unsigned neuron_id[ NEURON_ID_LEN ];
    unsigned id_string[ ID_STR_LEN ];
} NM_query_id_response;
```

The first byte of the request message specifies which nodes are to respond, whether unconfigured nodes, selected nodes, or nodes that are both selected and unconfigured. A node may be selected with the Respond to Query message. Optional fields may be present in the message which specify an address mode, a byte count and an array of bytes which must match a part of the destination node's memory. For interoperability, the length of the matched region can be up to 11 bytes long. The matched region may be in the read-only structure, the configuration structure, the SNVT structure or the application data area. The address mode specifies whether the address of the matched memory is an absolute address in the memory space of the NEURON CHIP, an address relative to the read-only data structure (see section A.1), or an address relative to the configuration data structure(see section A.6). The response message contains the 6-byte NEURON ID and the 8-byte program ID. Authentication is never used with this message.

Respond to Query

This message explicitly selects or deselects a node to respond to a Query ID message. It can be used to determine network topology. Resetting the node clears the selection.

Message declaration from NETMGMT.H:

```
typedef enum {
    MODE_OFF        = 0,
    MODE_ON         = 1,
} NM_respond_to_query_request;
```

The request message consists of a single byte specifying whether the node should be selected or deselected. Authentication is never used with this message.

Service Pin Message (Unsolicited)

This is an unsolicited message sent by a node when its service pin is grounded. It contains the node's NEURON ID followed by the program ID.

Message declaration from NETMGMT.H:

```
typedef struct {
    unsigned neuron_id[ NEURON_ID_LEN ];
    unsigned id_string[ ID_STR_LEN ];
} NM_service_pin_msg;
```

The message data contains the 6-byte unique NEURON ID assigned by the manufacturer of the NEURON CHIP, followed by the 8-byte ID of the application program in the NEURON CHIP. See section A.1.1 for details on these values. The service pin message is sent as a domain-wide broadcast on the domain whose length is zero. The source subnet and node IDs are both zero. Routers retransmit service pin messages on the domains in which they are configured.

B.1.2 Domain Table Messages

Update Domain

This message overwrites a domain table entry with a new value, and recomputes the configuration checksum. This assigns a domain, subnet and node identifier to the node, as well as an authentication key for that domain. The authentication key is transmitted in the clear, so that this message should not be used on an open network if authentication protection is desired. The node does not enter the configured state until a Set Node Mode message is sent to change its state to configured. The Update Domain message is honored even if the node is read/write protected. For a description of the domain table, see section A.2. If the Update Domain message is received by a node on the domain which is being updated, the response is returned in the new domain. A node that receives this message can take up to 330 msec to execute the function.

Message declaration from NETMGMT.H:

```
typedef struct {
    unsigned domain_index;
    unsigned id[ DOMAIN_ID_LEN ];
    unsigned subnet;
    unsigned must_be_one : 1; // this bit must be set to 1
    unsigned node        : 7;
    unsigned len;
    unsigned key[ AUTH_KEY_LEN ];
} NM_update_domain_request;
```

The request message consists of an index into the domain table (0 or 1), followed by an image of the domain table entry to be written in the format of a `domain_struct` declared in \LB\INCLUDE\ACCESS.H. The most significant bit of the byte containing the node ID must be set.

Query Domain (Request/Response only)

This message retrieves an entry in the domain table. This message is honored even if the node is read/write protected. For a description of the domain table, see section A.2.

Message declarations from NETMGMT.H:

```
typedef unsigned /* domain_index */ NM_query_domain_request;
typedef struct {
    unsigned id[ DOMAIN_ID_LEN ];
    unsigned subnet;
    unsigned      : 1;
    unsigned node : 7;
    unsigned len;
    unsigned key[ AUTH_KEY_LEN ];
} NM_query_domain_response;
```

The request message consists of an index into the domain table (0 or 1). The response message contains an image of the domain table entry that was read in the format of a domain_struct declared in \LB\INCLUDE\ACCESS.H.

Leave Domain

This message deletes a domain table entry, and recomputes the configuration checksum. After the message is processed, if the node does not belong to any domain, it becomes unconfigured and is reset. This message is honored even if the node is read/write protected. For a description of the domain table, see section A.2. If the Leave Domain message is received by a node on the domain which is to be left, no response is returned. If the message causes the node to leave the last domain it is configured in, its state becomes unconfigured. A node that receives this message can take up to 330 msec to execute the function.

Message declaration from NETMGMT.H:

```
typedef unsigned /* domain_index */ NM_leave_domain_request;
```

The request message consists of an index into the domain table (0 or 1).

Update Key

This message adds an increment to the current encryption key in a domain table entry to form a new key, and recomputes the configuration checksum. This allows re-keying of a node without having to transmit the new key in the clear on the network. This message is honored even if the node is read/write protected. For a description of the domain table, see section A.2. Senders of this message should be especially careful that the message is not received more than once, since its function is incremental. A node that receives this message can take up to 150 msec to execute the function.

Message declaration from NETMGMT.H:

```
typedef struct {
    unsigned domain_index;
    unsigned key[ AUTH_KEY_LEN ];
} NM_update_key_request;
```

The request message consists of an index into the domain table (0 or 1), followed by six bytes of authentication key increment.

B.1.3 Address Table Messages

Update Address

This message overwrites an address table entry with a new value, and recomputes the configuration checksum. An address table entry allows the node to implicitly address another node, or join a group. This message is honored even if the node is read/write

protected. For a description of the address table, see section A.3. A node that receives this message can take up to 130 msec to execute the function.

Message declaration from NETMGMT.H:

```
typedef struct {
    unsigned addr_index;
    unsigned type;                  // addr_type or (0x80 | group_size)
    unsigned domain           : 1;
    unsigned member_or_node   : 7;
    unsigned rpt_timer        : 4;
    unsigned retry            : 4;
    unsigned rcv_timer        : 4;
    unsigned tx_timer         : 4;
    unsigned group_or_subnet;
} NM_update_addr_request;
```

The request message consists of an index into the address table (0 to 14), followed by an image of the address table entry to be written, in the format of an address_struct declared in \LB\INCLUDE\ACCESS.H. The address type field is 0 for unbound, 1 for subnet_node, 3 for broadcast, and for group addressing it contains the group size with the most significant bit set.

Query Address (Request/Response only)

This message retrieves an entry in the address table. This message is honored even if the node is read/write protected. For a description of the address table, see section A.3.

Message declarations from NETMGMT.H:

```
typedef unsigned /* addr_index */   NM_query_addr_request;
typedef struct {
    unsigned type;                  // addr_type or (0x80 | group_size)
    unsigned domain           : 1;
    unsigned member_or_node   : 7;
    unsigned rpt_timer        : 4;
    unsigned retry            : 4;
    unsigned rcv_timer        : 4;
    unsigned tx_timer         : 4;
    unsigned group_or_subnet;
} NM_query_addr_response;
```

The request message consists of an index into the address table (0 to 14). The response message contains an image of the address table entry that was read in the format of an address_struct declared in \LB\INCLUDE\ACCESS.H. The address type field is 0 for unbound, 1 for subnet_node, 3 for broadcast, and for group addressing it contains the group size with the most significant bit set.

Update Group Address Data

This message updates a group entry in the address table with a new group size and timer fields, and recomputes the configuration checksum. The message is sent to all members of the group, and updates the corresponding entry in the address table. This is used when nodes join or leave the group. This message is honored even if the node is read/write protected. For a description of the address table, see section A.3. A node that receives this message can take up to 130 msec to execute the function.

Message declaration from NETMGMT.H:

```
typedef struct {
    unsigned type        : 1;  // must be one
    unsigned size        : 7;
    unsigned domain      : 1;
    unsigned member      : 7;
    unsigned rpt_timer   : 4;
    unsigned retry       : 4;
    unsigned rcv_timer   : 4;
    unsigned tx_timer    : 4;
    unsigned group;
} NM_update_group_addr_request;
```

The request message consists of an image of the address table entry to be written, and must be delivered with group addressing. The group size and timer values are updated with new values, but the member number and domain index are left unchanged. The group number must not change.

B.1.4 Network Variable-Related Messages

Network variable update and poll messages are described in section B.3.

Update Net Variable Config

This message overwrites a network variable configuration table entry with a new value, and recomputes the configuration checksum. This assigns a network variable selector to effect the binding of the network variable to network variables with the same selector on other nodes. This message is honored even if the node is read/write protected. For a description of the network variable configuration table, see section A.4. For a node running the Microprocessor Interface Program with the network variable configuration table on the host, the Update Net Variable Config message is passed to the host microprocessor - in this case, the network variable index value of 255 is reserved to indicate that the following two bytes in the message form a 16-bit network variable index. Only values 0 - 0xFFF are valid network variable indices, allowing up to 4,096 network variable table entries to be updated. A node that receives this message can take up to 90 msec to execute the function.

Message declaration from `NETMGMT.H`:

```
typedef struct {
    unsigned nv_index;
    unsigned nv_priority      : 1;
    unsigned nv_direction     : 1;
    unsigned nv_selector_hi   : 6;
    unsigned nv_selector_lo   : 8;
    unsigned nv_turnaround    : 1;
    unsigned nv_service       : 2;
    unsigned nv_auth          : 1;
    unsigned nv_addr_index    : 4;
} NM_update_nv_cnfg_request;
```

The request message consists of an index into the network variable table, followed by an image of the network variable configuration table entry to be written, in the format of an `nv_struct` declared in `\LB\INCLUDE\ACCESS.H`.

Query Net Variable Config (Request/Response only)

This message retrieves an entry in the network variable configuration table. This message is honored even if the node is read/write protected. For a description of the network variable configuration table, see section A.4. For a node running the Microprocessor Interface Program with the network variable configuration table on the host, the Query Net Variable Config message is passed to the host microprocessor — in this case, the network variable index value of 255 is reserved to indicate that the

following two bytes in the message form a 16-bit network variable index. Only values 0 - 0xFFF are valid, allowing up to 4,096 network variable configuration table entries to be queried.

Message declaration from NETMGMT.H:

```
typedef unsigned /* nv_index */     NM_query_nv_cnfg_request;
typedef struct {
    unsigned nv_priority      : 1;
    unsigned nv_direction     : 1;
    unsigned nv_selector_hi   : 6;
    unsigned nv_selector_lo   : 8;
    unsigned nv_turnaround    : 1;
    unsigned nv_service       : 2;
    unsigned nv_auth          : 1;
    unsigned nv_addr_index    : 4;
} NM_query_nv_cnfg_response;
```

The request message consists of an index into the network variable configuration table. The response message contains an image of the network variable configuration table entry that was read in the format of an `nv_struct` declared in `\LB\INCLUDE\ACCESS.H`.

Query SNVT (Request/Response only)

This message retrieves self-identification and self-documentation data from the host processor memory of a node running the Microprocessor Interface Program. This program is a special application used to attach the node to a host processor at Layer 5 or Layer 6 of the protocol. In this case, the address table is located in the NEURON CHIP memory, but the network variable fixed table and SNVT information is located in the host's memory. The specified amount of data is retrieved from the specified offset into the SNVT Structure on the host. The number of bytes read in one message is limited by the network buffer sizes on both NEURON CHIPs. For interoperability, this should be limited to 16 bytes. For a NEURON CHIP running a regular application program, the Query SNVT message will return the failed response. In this case the Read Memory message should be used to retrieve the SNVT information using the `snvt` pointer in the Read-only Structure (section A.1). For a description of the SNVT Structure, see section A.5.

Message declarations from NETMGMT.H:

```
typedef struct {
    unsigned long offset;
    unsigned count;
} NM_query_SNVT_request;
typedef unsigned NM_query_SNVT_response[];
```

The request message consists of two bytes specifying the offset into the addressed memory, and a byte specifying the number of bytes to be retrieved. The address is passed with the most significant byte first, whether or not this is the native address format of the host microprocessor. The response message contains the data in the memory that was read.

Network Variable Fetch (Request/Response only)

This message retrieves the value of a network variable by its index into the network variable tables. This can be used to poll the value of a network variable, even if the node is off-line. For a description of the network variable tables, see section A.3. The normal way to poll a network variable value is with an application message specifying the network variable's selector (see section B.3). For a node running the Microprocessor Interface Program, the Network Variable Fetch message is passed to the host microprocessor — in this case, the network variable index value of 255 is reserved to indicate that the following two bytes in the message form a 16-bit network variable

index. Only values 0 - 0xFFF are valid, allowing up to 4,096 network variables to be fetched.

Message declarations from NETMGMT.H:

```
typedef unsigned /* nv_index */ NM_NV_fetch_request;
typedef struct {
    unsigned nv_index;
    unsigned data[];
} NM_NV_fetch_response;
```

The request message consists of the network variable index, which is the index into either of the network variable tables. The response message contains the network variable index, followed by the network variable data itself.

B.1.5 Memory-Related Messages

Read Memory (Request/Response only)

This message reads data from the node's memory. Addresses may be specified relative to the Read-only Structure (section A.1), relative to the Configuration Structure (section A.6), or absolutely. To read the domain table, the address table, or the network variable configuration table, the Query Domain/Address/NV Config messages should be used. The number of bytes that can be be read in one message is limited only by the network buffer sizes on both NEURON CHIPs. For interoperability, this should be limited to 16 bytes. If the node is read/write protected, none of the node's memory may be read, except for the Read-only Structure (section A.1), the SNVT Structures (section A.5) and the Configuration Structure (Section A.6).

Message declarations from NETMGMT.H:

```
typedef struct {
    enum {
        ABSOLUTE            = 0,
        READ_ONLY_RELATIVE  = 1,
        CONFIG_RELATIVE     = 2,
    } mode;
    unsigned long offset;
    unsigned count;
} NM_read_memory_request;
typedef unsigned NM_read_memory_response[];
```

The request message consists of a byte specifying the address mode, two bytes specifying the offset into the addressed memory (most significant byte of the address first), and a byte specifying the number of bytes to be read. The response message contains the data in the memory that was read.

Write Memory

This message writes data to the node's memory. Addresses may be specified relative to the Read-only Structure (section A.1), relative to the Configuration Structure (section A.6), or absolutely. This message may be used to download an application program to read/write memory on the node. The number of bytes that can be written in one message is limited by the network buffer sizes on both NEURON CHIPs. For interoperability, this should be limited to 11 bytes. If writing to EEPROM, the application checksum and the configuration checksum may be recalculated. In this case, the number of bytes written in one message should be limited to 38 to avoid watchdog timeouts at maximum input clock rate. The node may also be reset. If the node is read/write protected, none of the node's memory may be written except for the Configuration Structure. To write the domain table, the address table, or the network variable configuration table, the Update Domain/Address/NV Config messages should be used.

Message declaration from NETMGMT.H:

```
typedef struct {
    enum {
        ABSOLUTE            = 0,
        READ_ONLY_RELATIVE  = 1,
        CONFIG_RELATIVE     = 2,
    } mode;
    unsigned long offset;
    unsigned count;
    enum {
        NO_ACTION               = 0,
        BOTH_CS_RECALC          = 1,
        CNFG_CS_RECALC          = 4,
        ONLY_RESET              = 8,
        BOTH_CS_RECALC_RESET    = 9,
        CNFG_CS_RECALC_RESET    = 12,
    } form;
    unsigned data[];
} NM_write_memory_request;
```

The request message consists of a byte specifying the address mode, two bytes specifying the offset into the addressed memory (most significant byte of the address first), a byte specifying the number of bytes to be written, and a byte specifying the action to be taken at the completion of the write operation, followed by the data bytes to be written.

Checksum Recalculate

This message recomputes either the network image checksum, or both the network and application image checksums in EEPROM. The application image and the network image are independently checked whenever the node is reset. They are also checked by a continually running background task. If the configuration checksum is invalid, the node enters the unconfigured state. If the application checksum is invalid, the node enters the application-less state. The network variable messages to update the domain, address or network variable tables automatically update the configuration checksum. The Checksum Recalculate message is intended for use after a series of Write Memory messages, for example, when downloading an application program.

Message declaration from NETMGMT.H:

```
typedef enum {
    BOTH_CS = 1,  // Application image and network image checksums
    CNFG_CS = 4,  // Network image checksum only
} nm_checksum_recalc;
```

The request message consists of a single byte specifying which checksums should be recalculated.

Memory Refresh

This message rewrites EEPROM memory at the specified offset. Either on-chip or off-chip EEPROM may be refreshed. The number of bytes refreshed in one message should be limited to 38 to avoid watchdog timeouts at maximum input clock rate. This message may be used periodically to extend the 10-year data retention of most EEPROM devices. Note that most EEPROM devices are specified as supporting 10,000 write cycles, therefore this message should not be used very frequently. The 48-bit NEURON ID cannot be refreshed. The Memory Refresh message returns the failed response if the address specified is outside of the on-chip or off-chip EEPROM regions.

Message declaration from NETMGMT.H

```
typedef struct {
    unsigned long offset;
    unsigned count;
    enum {
        ON_CHIP   = 0,
```

```
        OFF_CHIP  = 1,
    } which;
} NM_memory_refresh_request;
```

The request message consists of two bytes specifying the offset into the addressed memory (most significant byte of the address first), a byte specifying the number of bytes to be refreshed, and a byte indicating whether on-chip or off-chip EEPROM is to be refreshed.

B.1.6 Special-Purpose Messages

Set Node Mode (Service class varies)

This message puts the application in an off-line or on-line mode, or resets the node. This message may also be used to change the state of the node. The state may be:

Node State	State Code	Service LED
Application-less and unconfigured	3	On
Unconfigured (but with an application)	2	Flashing
Configured, hard off-line	6	Off
Configured, soft off-line	1 2	Off
Configured, on-line	4	Off

A node in the soft off-line state will go on-line when it is reset. The hard off-line state is preserved across a reset. When the application is off-line, the scheduler is disabled. Polling a network variable will return no data, but network variable updates will be processed. However, if the application is off-line, `nv_update_occurs` events will be lost. If this message changes the mode to off-line or on-line, the appropriate NEURON C task (if any) will be executed. *Mode on-line* and *mode off-line* messages should not be delivered with Request/Response service. There will be no response to a *Reset* message, since the node is reset immediately. Changing the state of a node recomputes the configuration checksum which takes some time, so verification of state changes should always be made with the Query Status message.

Message declaration from `NETMGMT.H`:

```
typedef struct {
    enum {
        APPL_OFFLINE = 0,      // soft offline state
        APPL_ONLINE  = 1,
        APPL_RESET   = 2,
        CHANGE_STATE = 3,
    } mode;
    enum {
        APPL_UNCNFG      = 2,
        NO_APPL_UNCNFG   = 3,
        CNFG_ONLINE      = 4,
        CNFG_OFFLINE     = 6,  // hard offline state
        SOFT_OFFINE      = 12,
    } state;                   // Optional field if mode = 3
} NM_set_node_mode_request;
```

The request message consists of a byte specifying whether this is a request to put the node in the soft offline state, put it online, reset it, or change the state of the NEURON CHIP. In this last case, there is a second byte specifying which state the node should enter.

Wink (Any service class except Request/Response)

This message causes the node to execute the *wink* clause in application program (if any). This may be used to identify nodes visually or audibly if it is more convenient than grounding the service pin. There is no data associated with this message.

B.2 Network Diagnostic Messages

Query Status (Request/Response only)

This message retrieves the network error statistics accumulators, the cause of the last reset, the state of the node, and the last run-time error logged. This message is used after a node has been reset to verify that the reset has occurred, since resets are not acknowledged.

Message declaration from NETMGMT.H:

```
typedef struct {
    unsigned long xmit_errors;
    unsigned long transaction_timeouts;
    unsigned long rcv_transaction_full;
    unsigned long lost_msgs;
    unsigned long missed_msgs;
    unsigned reset_cause;
    enum {
        APPL_UNCNFG      = 2,
        NO_APPL_UNCNFG   = 3,
        CNFG_ONLINE      = 4,
        CNFG_OFFLINE     = 6,          // hard offline state
        SOFT_OFFLINE     = 0xC,
        CNFG_BYPASS      = 0x8C,
    } node_state;
    unsigned version_number;
    enum {
        NO_ERROR                        = 0,
        BAD_EVENT                       = 1,
        NV_LENGTH_MISMATCH              = 2,
        NV_MSG_TOO_SHORT                = 3,
        EEPROM_WRITE_FAIL               = 4,
        BAD_ADDRESS_TYPE                = 5,
        PREEMPTION_MODE_TIMEOUT         = 6,
        ALREADY_PREEMPTED               = 7,
        SYNC_NV_UPDATE_LOST             = 8,
        INVALID_RESP_ALLOC              = 9,
        INVALID_DOMAIN                  = 10,
        READ_PAST_END_OF_MSG            = 11,
        WRITE_PAST_END_OF_MSG           = 12,
        INVALID_ADDR_TABLE_INDEX        = 13,
        INCOMPLETE_MSG                  = 14,
        NV_UPDATE_ON_OUTPUT_NV          = 15,
        NO_MSG_AVAIL                    = 16,
        ILLEGAL_SEND                    = 17,
        UNKNOWN_PDU                     = 18,
        INVALID_NV_INDEX                = 19,
        DIVIDE_BY_ZERO                  = 20,
        INVALID_APPL_ERROR              = 21,
        MEMORY_ALLOC_FAILURE            = 22,
        WRITE_PAST_END_OF_NET_BUFFER    = 23,
        APPL_CS_ERROR                   = 24,
        CNFG_CS_ERROR                   = 25,
        INVALID_XCVR_REG_ADDR           = 26,
        XCVR_REG_TIMEOUT                = 27,
        WRITE_PAST_END_OF_APPL_BUFFER   = 28,
```

```
        IO_READY                    = 29,
        SELF_TEST_FAILED            = 30,
        SUBNET_ROUTER               = 31,
    } error_log;
    unsigned model_number;
} ND_query_status_response;
```

The request message contains no data. The response message begins with five 16-bit error statistics accumulators as follows:

Transmission errors - number of CRC errors detected during packet reception. These may be due to collisions or noise on the transceiver input.

Transaction timeouts - number of times that the node failed to receive expected acknowledgements or responses after retrying the configured number of times. These may be due to destination nodes being inaccessible on the network, transmission failures because of noise on the channel, or if any destination node has insufficient buffers or receive transaction records. When using Request/Response service or network variable polling, a transaction timeout can also occur if the destination node application program does not return to the scheduler frequently enough because responses are synchronized with the application tasks.

Receive transaction full errors - number of times that an incoming packet was discarded because there was no room in the transaction database. These may be due to excessively long receive timers (section A.3.11), or inadequate size of the transaction database.

Lost messages - number of times that an incoming packet was discarded because there was no application buffer available. These may be due to an application program being too slow to process incoming packets, to insufficient application buffers, or to excess traffic on the channel. If the incoming message is too large for the application buffer, an error is logged, but the lost message count is not incremented.

Missed messages - number of times that an incoming packet was discarded because there was no network buffer available. These may be due to excess traffic on the channel, to insufficient network buffers, or to the network buffers not being large enough to accept all packets on the channel, whether or not addressed to this node.

The response message also contains a byte with the cause of the last reset as follows (X = don't care):

Power-up reset	0bXXXXXXX1
External reset	0bXXXXXX10
Watchdog reset	0bXXXX1100
Software reset	0bXXX10100

This is followed by a byte containing the current state of the node. The state may be:

Node State	State Code	Service LED
Application-less and unconfigured	3	On
Unconfigured (but with an application)	2	Flashing
Configured, hard off-line	6	Off
Configured, soft off-line	1 2	Off
Configured, on-line	4	Off

A node in the soft off-line state will go on-line when it is reset. The hard off-line state is preserved across a reset. A node in the unconfigured state normally does not execute its application program. The next byte in the response message indicates the version number of the firmware executing on the target node. For the firmware distributed with LONBUILDER 2.0, this is 2; for LONBUILDER 2.1, this is 3; for custom system images, this is the range 128-254. The version number is followed by a byte indicating the reason for the last error detected by the firmware on the target node. Zero means that no error

has been detected since the last reset. For a description of the firmware errors, see the *NEURON C Programmer's Guide*, appendix E. The last byte is the NEURON CHIP model number: zero for a 3150, and 8 for a 3120 .

Clear Status

This message clears the network error statistics accumulators and the error log. The request message contains no data.

Proxy Command (Request/Response only)

This message requests the node to deliver a Query ID, Query Status or Query Transceiver Status message to another node. This can be used when the target node is out of earshot of the network management node. The response is also relayed back to the original sender.

Message declaration from NETMGMT.H:

```
typedef struct {
    enum {
        QUERY_ID                = 0,
        STATUS_REQUEST          = 1,
        XCVR_STATUS             = 2,
    } sub_command;
    unsigned type;              // addr_type or (0x80 | group_size)
    unsigned                    : 1;
    unsigned member_or_node     : 7;
    unsigned rpt_timer          : 4;
    unsigned retry              : 4;
    unsigned rcv_timer          : 4;
    unsigned tx_timer           : 4;
    unsigned group_or_subnet;
    unsigned neuron_id[ 6 ];        // for type = 2
} ND_proxy_request;
```

The request message contains a byte specifying which message is to be delivered by proxy, followed by a destination address in the format of a `msg_out_addr` declared in `\LB\INCLUDE\MSG_ADDR.H`. See section A.3 for a description of destination address formats. The proxy message is delivered on the domain on which it was received. The agent node must be configured in the domain in which the message is received. The address type field is 1 for `subnet_node`, 2 for `neuron_id`, 3 for `broadcast`, and for group addressing it contains the group size with the most significant bit set. The response message is the response appropriate to the specified status request. The node requesting the proxy service must take into account the response time through the agent when setting the transaction timer.

Query Transceiver Status (Request/Response only)

This message retrieves transceiver status registers. This is a group of seven registers implemented in special-purpose mode transceivers. Even if the transceiver implements fewer than seven registers, seven values are returned in the response (see section 6.3).

Message declaration from NETMGMT.H:

```
typedef struct {
    unsigned xcvr_params[ NUM_COMM_PARAMS ];
} ND_query_xcvr_response;
```

The request message contains no data. The response message contains seven bytes with the transceiver status register values.

B.3 Network Variable Messages

Network variable messages are of two types — network variable updates, and network variable polls. All network variable messages are implemented with message codes in the range 0x80 - 0xFF, that is, the most significant bit of the code is set.

A network variable update message is sent whenever an output network variable is updated by the application program, and the variable has been declared without the *polled* qualifier. These messages may be sent with Acknowledged, Unacknowledged, or Unacknowledged/ Repeated service class. Updating an output network variable that has been declared with the *polled* qualifier does not cause a network variable update to be sent. A network variable update message contains the selector of the network variable that was updated, along with the data value. When a network variable update message is addressed to a node that has an input network variable whose selector matches the network variable selector in the message, then a network variable update event occurs on the destination node, and the value of the input network variable is modified with the data in the message. For a node running the Microprocessor Interface Program, the matching of selectors may be done in the NEURON CHIP, or on the host processor

A network variable poll message is sent when the application program calls the *poll()* system function specifying one or all of its input network variables. This message is sent with request/response service, and contains the selector of the polled network variable. When a network variable poll message is addressed to a node which has an output network variable whose selector matches the network variable selector in the message, then that node responds with a message containing the data in the network variable. This response is treated the same as a network variable update message; a network variable update event occurs on the requesting node, and the value of the input network variable is modified with the data in the message.

Normally, network variable updates and polls are delivered using implicit addressing, namely using an entry in the address table of the source node as the destination address. However, for special applications, it is possible to explicitly address network variable updates and polls by sending explicit messages that have the same structure as network variable messages.

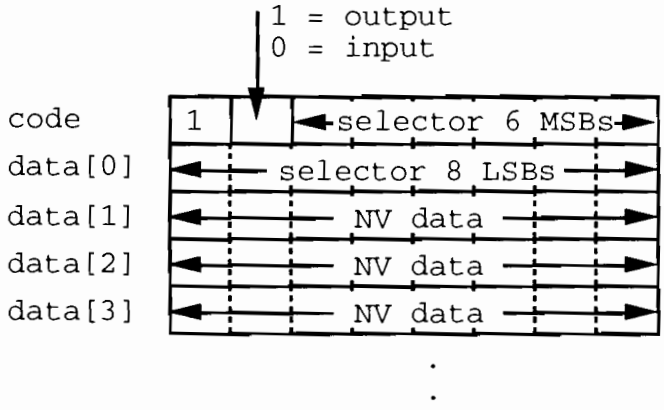


Figure B.1 Network Variable Message Structure

Network Variable Update (Acknowledged, Unacknowledged or Unacknowledged/Repeated)

The code field of this message contains the most significant six bits of the network variable selector. The most significant bit of the code field is set, indicating a network

variable message, and the second most significant bit is clear, indicating that the update is addressed to an input network variable. The first data byte of the message contains the least significant eight bits of the network variable selector, and this is followed by the data in the network variable itself. The length of the data in the message must match the length of the destination network variable.

Example of updating a two-byte network variable whose selector is `0x1234` with the data value `0x5678`:

```
msg_out.code = 0x80 | 0x12;    // code field = 0x80 | nv_selector_hi
msg_out.data[ 0 ] = 0x34;      // data field = nv_selector_lo
msg_out.data[ 1 ] = 0x56;      // high byte of network variable value
msg_out.data[ 2 ] = 0x78;      // low byte of network variable value
```

Note that a network variable update message is processed by the network processor on the destination node. In a normal application program, the network variable update cannot be received in the application processor using explicit messaging syntax. If an application needs to extract the source address from an incoming network variable update message, the NEURON C `nv_in_addr` built-in variable can be used. Nodes based on the Microprocessor Interface Program can optionally receive the network variable update on the host processor.

Network Variable Poll (Request/Response only)

For completeness, the network variable poll message format is described here, although this message is not very useful in a NEURON C application program. This is because the response to the poll will be processed by the network processor, and cannot be received in the application program using explicit messaging syntax. If the node sending the poll message has an input network variable with the same selector and the same size as the polled network variable, then this network variable will be updated by the response to the poll. A more convenient method of reading the value of a network variable with an explicitly addressed message is with the Network Variable Fetch network management message described in section B.2. Alternatively, a node running the Microprocessor Interface Program can handle the response to the poll explicitly on the host processor.

The code field of the Network Variable Poll request message contains the most significant six bits of the network variable selector. The most significant bit of the code is set, indicating a network variable message, and the second most significant bit is set indicating that the poll is addressed to an output network variable. The first data byte of the request message contains the least significant eight bits of the network variable selector. The response message contains the same code, except that the second most significant bit is clear, indicating that the response is addressed to an input network variable. The first data byte of the response contains the least significant eight bits of the network variable selector, and this is followed by the data in the network variable itself. If the poll is received by a node that has no matching network variable, or the node is offline, then the response contains the selector, but no data is present.

Disclaimer

NEURON CHIPs are manufactured and sold under license from Echelon by Motorola Inc. and Toshiba Corporation. Please obtain current data sheets from these manufacturers. All specifications contained herein are subject to change without notice.

Echelon Corporation assumes no responsibility for any errors contained herein. Echelon makes no representation and offers no warranty of any kind regarding any of the third-party components mentioned in this document. These components are suggested only as examples of usable devices. The use of these components or other alternatives is at the customer's sole discretion. Echelon also does not guarantee the designs shown in this document.

NEURON CHIPs were not designed for use in equipment or systems which involve danger to human health or safety or a risk of property damage and Echelon assumes no responsibility or liability for use of the NEURON CHIPs in such applications.

In order to purchase NEURON CHIPs, customers must first enter into a Developer's License Agreement or an OEM License Agreement.

For more information, please call 1-800-258-4LON, or 415-855-7400 No part of this document may be reproduced, translated, or transmitted in any form without permission from Echelon.

Part Number 005-0018-01
Rev. C

Echelon Corporation
4015 Miranda Avenue
Palo Alto, CA 94304
Telephone (415) 855-7400
Fax (415) 856-6153

Echelon Europe Ltd
105 Heath Street
London NW3 6SS
England
Telephone (071) 431-1600
Fax (071) 794-0532
International Telephone + 44 71 431-1600
International Fax + 44 71 794-0532

Echelon Japan K.K.
AIOS Gotanda Building #808
10-7, Higashi-Gotanda 1-chome,
Shinagawa-ku, Tokyo 141, Japan
Telephone (03) 3440-8638
Fax (03) 3440-8639